Dossier d'Architecture Technique (DAT)

Plateforme CRM SaaS Multi-Tenant

Conception de l'architecture technique d'une plateforme CRM (Customer Relationship Management) en mode SaaS, intégrant des exigences de sécurité, de scalabilité, de disponibilité et de maintenabilité.

Ce document présente l'ensemble des choix architecturaux, techniques et organisationnels relatifs à la conception du système CRM SaaS.

Réalisé par :

- **Mahmoudi Sarra**
- **Leveneur Yannis**
- **Pasqualini Antoine**

Établissement :

Paris School of Technology & Business (PSTB)

Année universitaire :

2025 – 2026

Sommaire

0. Introduction & Contexte du Projet

0.1 Présentation du projet

Ce projet consiste à concevoir l'architecture technique d'une plateforme CRM (Customer Relationship Management) en mode SaaS, destinée à des entreprises clientes souhaitant gérer efficacement leurs relations commerciales.

La plateforme permettra de centraliser et structurer les informations liées aux clients, prospects, opportunités commerciales et interactions, tout en offrant des fonctionnalités de segmentation, reporting et intégration avec des services externes (email, calendrier, outils marketing).

Le système est conçu comme une application multi-tenant, où plusieurs entreprises (tenants) utilisent la même plateforme tout en garantissant une isolation stricte des données et une sécurité élevée

L'objectif principal de ce projet n'est pas le développement du logiciel, mais la **conception d'une architecture technique robuste, sécurisée, scalable et maintenable**, conforme aux bonnes pratiques vues dans le cours d'architecture des systèmes d'information.

0.2 Objectifs business du système

Les objectifs business de la plateforme CRM sont les suivants :

- Améliorer la gestion de la relation client pour les entreprises utilisatrices
- Centraliser les données clients et commerciales dans un système unique
- Faciliter le suivi des opportunités et du cycle de vente
- Offrir des capacités de reporting et d'analyse pour la prise de décision
- Proposer une solution accessible en ligne, sans installation locale
- Permettre une montée en charge progressive en fonction de la croissance des clients
- Garantir un haut niveau de disponibilité et de fiabilité du service

D'un point de vue économique, la plateforme vise un modèle SaaS par abonnement, avec des entreprises clientes de tailles variables, impliquant des volumétries et usages hétérogènes.

0.3 Problématique à résoudre

La conception de cette plateforme CRM pose plusieurs problématiques majeures :

- Comment héberger un système partagé entre plusieurs entreprises tout en garantissant l'isolation des données ?
- Comment assurer la scalabilité face à des clients ayant des besoins très différents ?

- Comment maintenir de bonnes performances malgré des recherches, filtres et requêtes complexes ?

- Comment sécuriser les données sensibles des clients (informations commerciales, données personnelles) ?
- Comment concevoir une architecture évolutive, capable d'intégrer de nouvelles fonctionnalités sans refonte majeure ?
- Comment garantir une disponibilité élevée dans un contexte SaaS ?
- Ces problématiques justifient une réflexion architecturale approfondie, en particulier sur les choix d'infrastructure, de communication applicative, de sécurité et de structuration du système

0.4 Public cible

La plateforme CRM s'adresse aux publics suivants :

Entreprises clientes (tenants)

•PME et entreprises de taille intermédiaire

•Équipes commerciales, marketing et service client

•Organisations souhaitant externaliser leur CRM via une solution SaaS

Utilisateurs finaux

•Commerciaux

•Responsables commerciaux

•Managers

•Administrateurs fonctionnels du CRM

Administrateurs techniques

•Équipe interne exploitant et maintenant la plateforme

•Équipe DevOps / Sécurité

0.5 Périmètre fonctionnel

Fonctionnalités incluses dans le périmètre

•Gestion des entreprises clientes (multi-tenant)

•Gestion des utilisateurs et des rôles

•Gestion des contacts et prospects

•Gestion des opportunités commerciales

•Historique des interactions (emails, appels, notes)

•Segmentation des clients

•Tableaux de bord et reporting

•Intégration avec des services externes (email, calendrier)

•Authentification et autorisation sécurisées

Fonctionnalités hors périmètre

•Facturation et paiement des abonnements

•Intelligence artificielle avancée (prédictions, scoring automatisé)

•Téléphonie intégrée

•Développement mobile natif

Ces éléments sont volontairement exclus afin de contenir la complexité du projet et de se concentrer sur l'architecture technique cœur.

0.6 Contraintes du projet

Contraintes techniques

•Architecture orientée services

•Communication sécurisée via HTTPS

•Support de la montée en charge horizontale

•Haute disponibilité et tolérance aux pannes

•Compatibilité avec des intégrations externes

Contraintes de sécurité

•Isolation stricte des données entre tenants

•Authentification forte des utilisateurs

•Autorisation basée sur les rôles (RBAC)

•Chiffrement des données en transit

•Journalisation et traçabilité des accès

Contraintes de budget et de temps

- Choix technologiques réalistes et cohérents
- Architecture concevable et justifiable sans sur-ingénierie

Contraintes réglementaires

*Conformité au **RGPD** :*

  - Protection des données personnelles
  - Minimisation des données
  - Traçabilité des accès
  - Possibilité de suppression des données

- Localisation et gestion responsable des données clients

Diagramme C4

Niveau 1 : Context

Le diagramme de contexte présente une vue d'ensemble du système CRM SaaS et de son environnement. Il permet de répondre à la question :
« Quel est le périmètre du système et avec quels acteurs et systèmes externes interagit-il ? »

La plateforme CRM SaaS est utilisée par des entreprises clientes afin de gérer leurs relations clients, leurs opportunités commerciales et leurs interactions.
Le système interagit également avec plusieurs services externes nécessaires à son fonctionnement, tels que les services d'email, de calendrier et d'authentification.

Acteurs

- **Utilisateur Métier** : utilise le CRM pour gérer les contacts, opportunités et activités commerciales.
- **Administrateur Client** : gère les utilisateurs, les rôles et les paramètres de l'entreprise cliente.
- **Administrateur Technique** : assure la supervision, la maintenance et la sécurité de la plateforme.

Systèmes externes

- **Service Email Externe** : envoi d'emails et de notifications.
- **Service de Calendrier Externe** : synchronisation des rendez-vous et événements.
- **Service d'Authentification Externe** : gestion de l'authentification et du Single Sign-On

Interactions principales

- Les utilisateurs et administrateurs accèdent à la plateforme CRM SaaS.
- La plateforme CRM communique avec les services externes pour l'email, le calendrier et l'authentification.

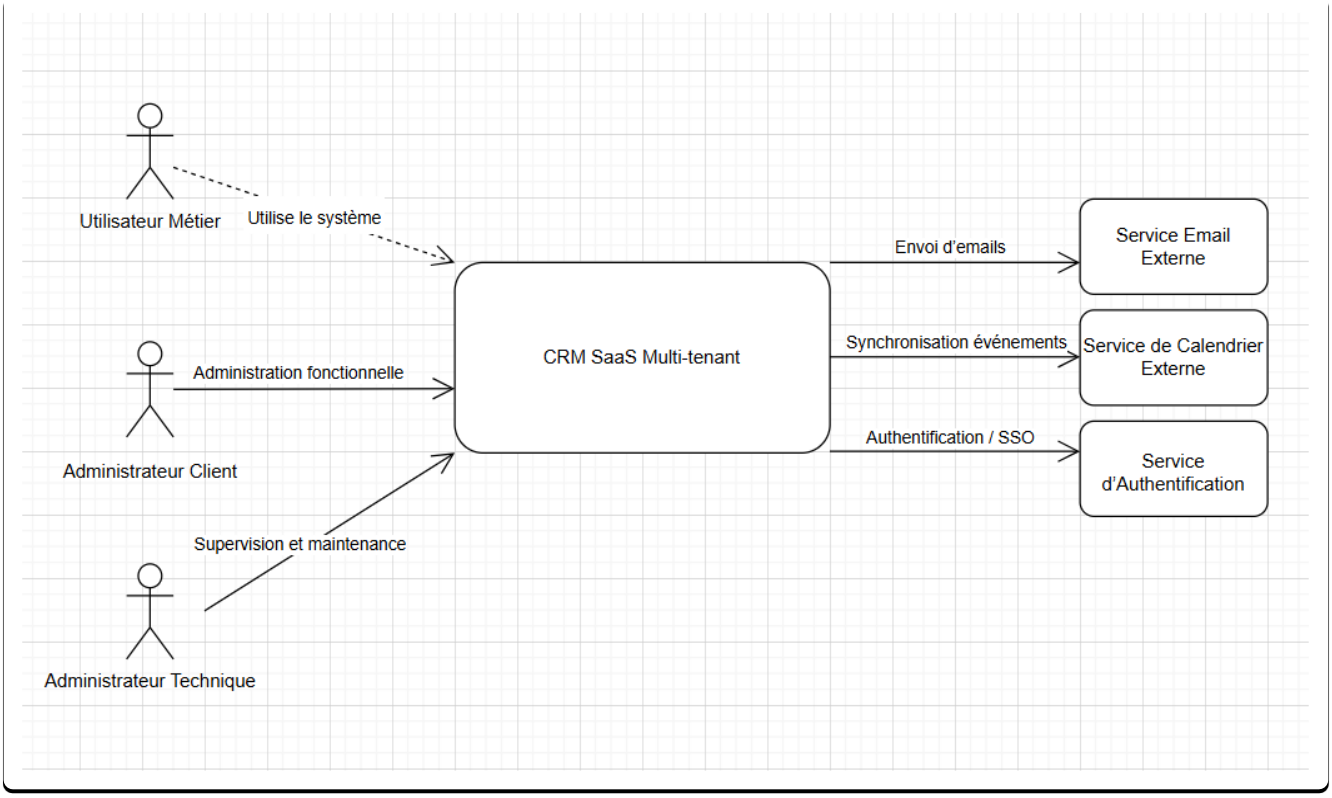


Diagramme C4 – Niveau 2 : Containers

Objectif

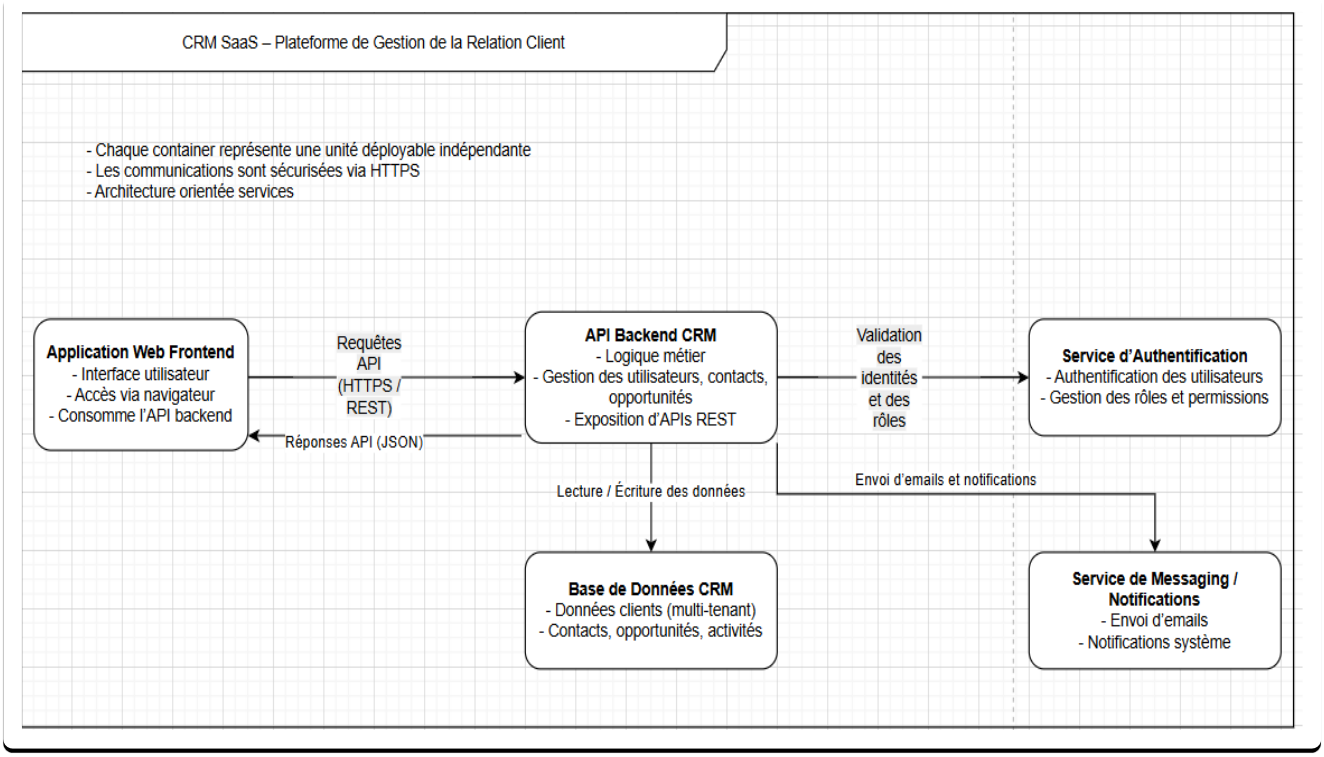
Le diagramme de containers décrit les principales unités déployables du système et leurs interactions. Chaque container représente une application, un service ou une base de données pouvant être déployé indépendamment.

Description

L'architecture du CRM SaaS est organisée en plusieurs containers distincts afin d'assurer la séparation des responsabilités, la scalabilité et la sécurité du système.

Les principaux containers sont :

- Une application frontend accessible via navigateur.
- Une API backend exposant les fonctionnalités métier.
- Une base de données centrale.
- Des services transverses assurant la sécurité, la communication et l'observabilité.



Cette approche répond directement aux exigences non fonctionnelles du projet, notamment:

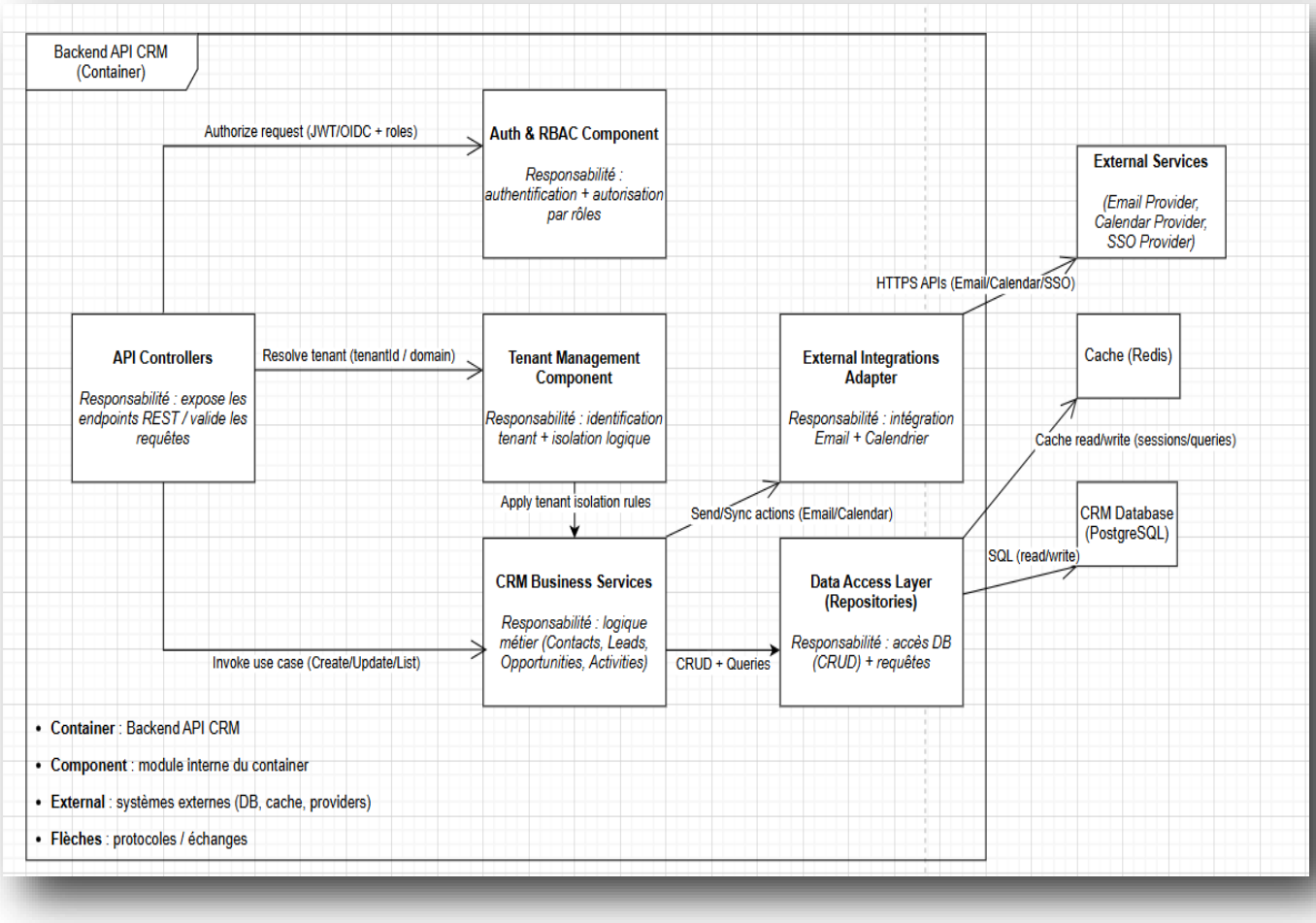
- **Scalabilité** : chaque container peut évoluer indépendamment selon la charge (frontend, backend, services transverses).
- **Sécurité** : les accès sont strictement contrôlés entre les containers, en particulier l'accès à la base de données, limité à l'API backend.
- **Maintenabilité** : la séparation des responsabilités réduit le couplage et facilite l'ajout de nouvelles fonctionnalités.
- **Résilience** : la défaillance d'un container (ex. service de notification) n'entraîne pas l'arrêt complet du système.
- **Évolutivité** : cette organisation permet, à terme, une transition progressive vers une architecture plus distribuée si nécessaire.

Le diagramme C4 – Niveau Containers constitue ainsi une vue intermédiaire essentielle entre la vision globale du système (diagramme de contexte) et le détail interne des composants applicatifs, qui sera présenté dans le diagramme de niveau Components.

Niveau 3 : Components

Le diagramme C4 – Niveau 3 vise à décrire l'organisation interne d'un container critique, en identifiant les composants logiciels majeurs, leurs responsabilités et leurs interactions.

Dans notre projet CRM SaaS, le container **le plus critique** est :
l'API Backend CRM



Détail interne du container critique (Backend API CRM)

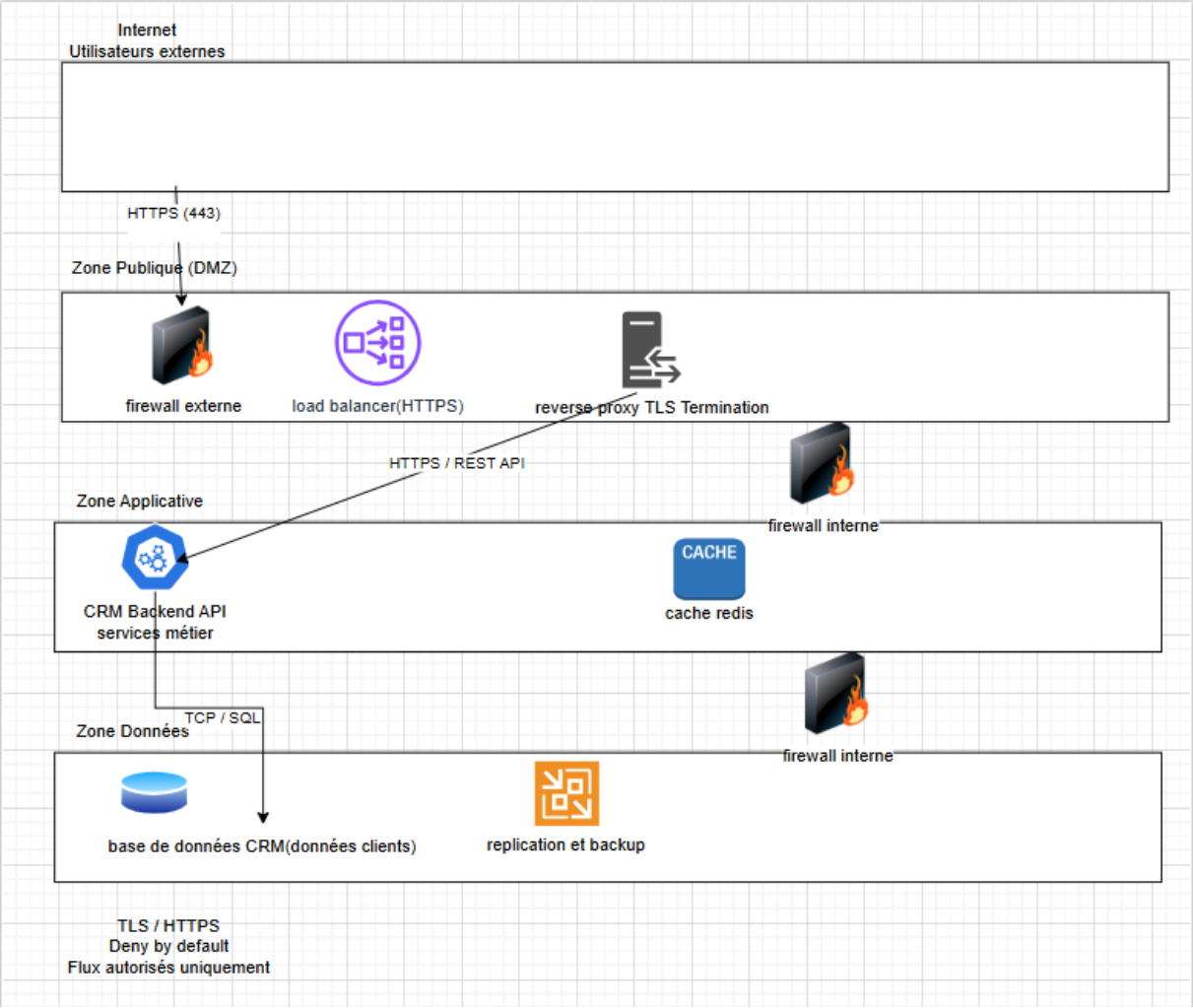
Le Backend API CRM est structuré en composants afin d'assurer la séparation des responsabilités et la maintenabilité :

- Les **API Controllers** exposent les endpoints REST et délèguent la logique métier.
- Le composant **Auth & RBAC** gère l'authentification (JWT/OIDC) et l'autorisation par rôles.
- Le composant **Tenant Management** résout le tenant et applique l'isolation multi-tenant.
- Les **CRM Business Services** implémentent les cas d'usage métier (contacts, leads, opportunités, activités).
- La couche **Repositories** centralise l'accès aux données (PostgreSQL) et au cache (Redis).
- Le composant **External Integrations Adapter** gère les appels vers les services externes (email, calendrier, SSO).

SCHÉMA RÉSEAU & SÉCURITÉ

Le schéma réseau montre où vivent techniquement les composants du système, comment ils communiquent, et comment ils sont protégés.

Comment les utilisateurs accèdent au système et comment le réseau empêche une attaque de tout casser ?



Firewall : filtrage réseau

Load Balancer : répartition de charge

TLS Termination : chiffrement HTTPS

API : services métier

DB : données sensibles

Le schéma réseau présente une architecture segmentée en zones de sécurité afin de respecter le principe du moindre privilège.
L'accès depuis Internet est limité à la zone publique (DMZ), exposant uniquement le point d'entrée HTTPS via un load balancer et un reverse proxy assurant la terminaison TLS.

La zone applicative héberge les services métier du CRM ainsi que les mécanismes de cache.
La zone données est strictement isolée et accessible uniquement depuis la zone applicative via des flux contrôlés, garantissant la protection des données clients.
Toutes les communications sont chiffrées et les flux non explicitement autorisés sont bloqués par défaut.

3. Choix d'Infrastructure

3.1 On-Premise vs Cloud vs Hybride

Options étudiées

Trois options d'infrastructure ont été analysées pour l'hébergement de la plateforme CRM SaaS :

Infrastructure On-Premise

Hébergement de l'ensemble du système dans les datacenters internes de l'entreprise.

Infrastructure Cloud

Hébergement sur une plateforme cloud publique (type AWS, Azure ou GCP), en utilisant des ressources virtualisées et des services managés.

Infrastructure Hybride

Combinaison d'une infrastructure on-premise pour certaines données ou services critiques et du cloud pour les composants applicatifs exposés ou scalables.

Option d'infrastructure	Avantages	Inconvénients
On-Premise	• Contrôle total sur l'infrastructure et les données • Conformité facilitée pour certaines contraintes réglementaires • Coûts prévisibles à long terme	• Investissement initial élevé (serveurs, réseau, sécurité) • Scalabilité limitée et lente • Maintenance et exploitation complexes • Peu adapté à un modèle SaaS évolutif
Cloud	• Élasticité et montée en charge rapide • Modèle économique *pay-as-you-go* • Services managés (load balancing, bases de données, monitoring) • Haute disponibilité native • Réduction de la charge opérationnelle • Adapté aux usages variables des tenants SaaS	• Dépendance au fournisseur (*vendor lock-in*) • Coûts variables pouvant augmenter avec la croissance • Gouvernance et sécurité à maîtriser
Hybride	• Migration progressive • Possibilité de conserver certaines données sensibles on-premise • Flexibilité architecturale	• Complexité élevée • Gestion de deux environnements distincts • Synchronisation des données plus difficile • Surcoût opérationnel

choix retenu et justification

Le choix retenu est une infrastructure Cloud.

Justification :

- Le projet vise une **plateforme CRM en mode SaaS**, nécessitant une forte **élasticité**.
- Les usages et volumétries peuvent varier fortement selon les entreprises clientes.
- Le cloud permet une **scalabilité horizontale**, une **haute disponibilité** et un **déploiement rapide**.
- Les services managés réduisent la complexité d'exploitation et facilitent la mise en œuvre des bonnes pratiques de sécurité et de monitoring.
- Ce choix est cohérent avec les architectures modernes étudiées dans le cours.

3.2 Dimensionnement

Le dimensionnement initial est basé sur des hypothèses raisonnables pour une première version de la plateforme, avec une montée en charge progressive.

Hypothèses de charge

- Plusieurs entreprises clientes (tenants)
- Quelques centaines à milliers d'utilisateurs actifs
- Accès simultanés variables selon les heures
- Requêtes fréquentes de lecture (consultation CRM)
- Écritures régulières (création d'opportunités, interactions)

Ressources estimées

Couche applicative (Backend CRM API)

- **CPU** : 2 à 4 vCPU par instance
- **RAM** : 4 à 8 Go
- **Instances multiples** pour la répartition de charge
- Architecture **stateless** pour permettre le scaling horizontal

Cache (Redis)

- **RAM** : 2 à 4 Go
- Utilisé pour :
    - Sessions
    - Données fréquemment consultées

- Réduction de la charge sur la base de données

Base de données CRM

- **CPU** : 4 à 8 vCPU
- **RAM** : 8 à 16 Go
- **Stockage** :
  - SSD
  - Volume initial adapté aux données clients
  - Extension possible via scaling vertical ou read replicas

Bande passante

- Accès utilisateurs via HTTPS
- Flux API internes sécurisés
- Dimensionnement prévu pour absorber des pics de trafic
- Utilisation du load balancer pour lisser la charge

3.3 Scalabilité et Haute Disponibilité

Scalabilité verticale

- Augmentation des ressources (CPU/RAM) des serveurs existants
- Utilisée ponctuellement, notamment pour la base de données

Scalabilité horizontale

- Ajout de nouvelles instances backend CRM
- Répartition automatique via le load balancer
- Nécessite des services **stateless**
- Recommandée pour une plateforme SaaS

Auto-scaling

- Déclenchement automatique basé sur :

    - Charge CPU
    - Nombre de requêtes
    - Latence
- Permet d'optimiser les coûts et la performance
- Réponse dynamique aux pics de charge

Haute disponibilité

- Plusieurs instances applicatives réparties
- Load balancer redondant
- Base de données avec réplication et sauvegardes régulières
- Tolérance aux pannes partielles sans interruption du service

4. Choix Applicatifs et Données

4.1 Stack Technique Applicative

Élément	Choix retenu	Justification (projet CRM SaaS)	Alternatives considérées (résumé)
Frontend (framework)	**React (SPA)**	Interfaces CRM riches (tableaux, filtres, formulaires), forte maturité, écosystème UI important, bonne maintenabilité.	Angular (plus "framework", plus lourd), Vue (plus simple mais écosystème entreprise parfois moins standardisé).
Backend (framework)	**Node.js + NestJS**	Structure "enterprise" (modules, DI), maintenabilité, productivité, adapté aux APIs REST, facile à industrialiser (tests, logs).	Express (trop libre, risque d'incohérence), Java Spring (robuste mais plus lourd), Python Django/DRF (très bon mais choix équipe/écosystème).
API (REST / GraphQL / gRPC)	**REST (API publique)**	Standard web, simple à documenter (OpenAPI), facile à consommer (front + intégrations externes). Suffisant pour une V1.	GraphQL (utile si besoins front très variables, mais plus complexe côté serveur), gRPC (excellent interne microservices, moins adapté en API publique).
Authentification (OAuth2 / OIDC / JWT)	**OIDC + OAuth2 + JWT**	OIDC pour "login/SSO" (identité), OAuth2 pour l'autorisation, JWT pour stateless et scalabilité (API Gateway/Backend).	Sessions serveur (moins scalable), API keys (moins adaptées pour utilisateurs finaux).
CI/CD	**GitLab CI (ou GitHub Actions)**	Pipelines automatisés (tests, build, scan, déploiement), traçabilité, reproductibilité, aligné bonnes pratiques DevOps.	Jenkins (puissant mais plus lourd à opérer), Azure DevOps (bien si écosystème Microsoft).
Gestion des secrets	**Vault (ou Secret Manager cloud)**	Centralisation + rotation + contrôle d'accès strict (moindre privilège). Évite secrets en clair dans le code/pipelines.	Variables CI simples (OK pour démo, moins robuste), stockage local (à éviter).

4.3 Services Transverses

Service transverse	Choix retenu	Rôle dans l'architecture	Alternatives / remarques

Messaging	**RabbitMQ**	Asynchrone pour emails, notifications, traitements “background”, découplage temporel, bonne latence.	Kafka si très gros volumes + besoin de replay/streaming massif.
Monitoring & logs	**Prometheus + Grafana (métriques) / ELK ou Loki (logs)**	Surveiller CPU/RAM/latence, erreurs, disponibilité; centraliser logs applicatifs & sécurité.	Datadog (SaaS), Cloud monitoring natif selon fournisseur.
Observabilité	**OpenTelemetry + Jaeger (traces)**	Suivre une requête à travers services, diagnostiquer latence/pannes, essentiel si architecture distribuée.	Zipkin possible.
Orchestration	**Docker** (base) + **Kubernetes** (si besoin scaling avancé)	Docker pour packaging et cohérence. Kubernetes si plusieurs services + auto-scaling + rolling updates.	Docker Compose (simple pour dev), K8s à éviter trop tôt si monolithe.

MODÈLE DE DONNÉES

5.1 Principe Multi-Tenant (commun à tout le modèle)

La plateforme CRM est **multi-tenant** : plusieurs entreprises clientes (tenants) partagent la même application, mais leurs données doivent rester **strictement isolées**.
Chaque entité métier contient donc un champ tenant_id (FK) qui rattache l’enregistrement à une entreprise donnée.

Règle d’isolation : toute requête (lecture/écriture) doit être filtrée par tenant_id afin d’empêcher l’accès aux données d’un autre tenant.

5.2 Classes / Entités (Schéma 1 — Périmètre fonctionnel)

A) Tenant

Rôle : représente une entreprise cliente utilisant la plateforme (un “client SaaS”).
Attributs (principaux) :

- tenant_id (PK) : identifiant unique du tenant
- name : nom de l’entreprise cliente
- status : état du tenant (actif / suspendu…)
- created_at : date de création

B) User

Rôle : utilisateur interne au tenant (commercial, manager, admin client).
Attributs :

- user_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- email
- password_hash
- first_name, last_name

- role (ADMIN / SALES / MANAGER) — RBAC simplifié
- is_active
- last_login_at
- created_at

C) Contact

Rôle : contact client/prospect géré dans le CRM (personne).
Attributs :

- contact_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- first_name, last_name
- email
- phone_main
- phone_alt (nullable)
- company_name (nullable)
- job_title (nullable)
- status
- created_at, updated_at

D) Lead

Rôle : prospect (lead) non converti, avant qualification en contact/opportunité.
Attributs :

- lead_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- first_name, last_name
- email (nullable)
- phone (nullable)
- source (nullable)
- status (new / qualified / lost)

- created_at, updated_at

E) Opportunity

Rôle : opportunité commerciale (deal) associée à un contact ou un lead.
Attributs :

- opportunity_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- owner_user_id (FK -> User.user_id)
- contact_id (FK -> Contact.contact_id, nullable)
- lead_id (FK -> Lead.lead_id, nullable)
- title
- stage (prospecting / negotiation / won / lost …)
- amount_estimated (nullable)
- currency (nullable)
- probability (nullable)
- expected_close_date (nullable)
- created_at, updated_at

Note de cohérence : une opportunité doit être reliée soit à un contact_id, soit à un lead_id (au moins un des deux non NULL).

F) Activity

Rôle : historique des interactions (email, appel, meeting, note).
Cette table assure la traçabilité des actions et constitue la base de l'historique CRM.
Attributs :

- activity_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- created_by_user_id (FK -> User.user_id)
- contact_id (FK -> Contact.contact_id, nullable)
- lead_id (FK -> Lead.lead_id, nullable)
- opportunity_id (FK -> Opportunity.opportunity_id, nullable)

- type (EMAIL / CALL / NOTE / MEETING)
- subject (nullable)
- content (nullable)
- direction (IN / OUT, nullable)
- activity_date
- created_at

Note : une activité peut être rattachée à un contact, un lead et/ou une opportunité selon le cas d'usage.

G) Tag (Segmentation)

Rôle : représente un label de segmentation (ex : "VIP", "Finance", "Cold Lead").
Attributs :

- tag_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- label
- color (nullable)
- created_at

H) ContactTag (association Contact ↔ Tag)

Rôle : table de liaison permettant d'associer plusieurs tags à un contact (many-to-many).
Attributs :

- contact_id (FK -> Contact.contact_id)
- tag_id (FK -> Tag.tag_id)
- assigned_at

Clé : PK composite recommandée (contact_id, tag_id) pour éviter les doublons.

5.3 Relations et cardinalités (Schéma 1)

Tenant (1) — (0..) User*

- Un tenant possède plusieurs utilisateurs.
- Un user appartient à un seul tenant.

Tenant (1) — (0..) Contact*

- Un tenant gère plusieurs contacts.
- Un contact appartient à un seul tenant.

Tenant (1) — (0..) Lead*

- Un tenant gère plusieurs leads.
- Un lead appartient à un seul tenant.

Tenant (1) — (0..) Opportunity*

- Un tenant gère plusieurs opportunités.
- Une opportunité appartient à un seul tenant.

User (1) — (0..) Opportunity (owner)*

- Un user peut posséder plusieurs opportunités.
- Une opportunité a un propriétaire (owner_user_id).

Contact (1) — (0..) Opportunity*

- Un contact peut avoir plusieurs opportunités.
- Une opportunité peut référencer un contact (contact_id nullable).

Lead (1) — (0..) Opportunity*

- Un lead peut générer plusieurs opportunités.
- Une opportunité peut référencer un lead (lead_id nullable).

User (1) — (0..) Activity (created_by)*

- Un user crée plusieurs activités.
- Une activité est créée par un seul user.

Contact (1) — (0..) Activity (optionnel)*

- Une activité peut concerner un contact (contact_id nullable).

Lead (1) — (0..) Activity (optionnel)*

- Une activité peut concerner un lead (lead_id nullable).

Opportunity (1) — (0..) Activity (optionnel)*

- Une activité peut concerner une opportunité (opportunity_id nullable).

Contact () — () Tag via ContactTag

- Un contact peut avoir plusieurs tags.
- Un tag peut être associé à plusieurs contacts.

5. Modèle de Données — Schéma 2 (Extensions hors périmètre)

5.4 Objectif du schéma 2

Le schéma 2 présente des entités **volontairement hors périmètre** (facturation, paiement, téléphonie, support/SLA), afin de démontrer l'évolutivité de l'architecture et la possibilité d'étendre le CRM sans refonte du cœur métier.

5.5 Classes / Entités ajoutées (Schéma 2)

A) Invoice (Facturation)

Rôle : gestion des factures d'abonnement (hors périmètre officiel).
Attributs :

- invoice_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- customer_contact_id (FK -> Contact.contact_id)
- invoice_number
- amount_total
- currency
- status (DRAFT / SENT / PAID / CANCELLED)
- issued_at
- due_date
- created_at

B) Payment (Paiement)

Rôle : enregistrement des paiements liés à une facture.
Attributs :

- payment_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- invoice_id (FK -> Invoice.invoice_id)
- amount
- payment_date
- status (PENDING / SUCCESS / FAILED)
- created_at

C) PaymentMethod (Mode de paiement)

Rôle : mode de paiement configuré par tenant (carte, virement, PayPal…).

Attributs :

- payment_method_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- type (CARD / BANK_TRANSFER / PAYPAL)
- provider
- is_default
- created_at

D) PhoneCall (Téléphonie intégrée)

Rôle : gestion avancée des appels (si téléphonie intégrée).

Attributs :

- call_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- contact_id (FK -> Contact.contact_id, nullable)
- user_id (FK -> User.user_id)
- direction (IN / OUT)
- started_at
- duration_seconds
- status (COMPLETED / MISSED)
- created_at

E) SupportTicket (Support client)

Rôle : tickets de support client (hors périmètre officiel).

Attributs :

- ticket_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- created_by_user_id (FK -> User.user_id)
- assigned_user_id (FK -> User.user_id, nullable)
- subject
- status (OPEN / IN_PROGRESS / CLOSED)

- priority (LOW / MEDIUM / HIGH)
- created_at
- closed_at (nullable)

F) SLA (Service Level Agreement)

Rôle : définition des engagements de support par tenant (temps de réponse / résolution).
Attributs :

- sla_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- name
- response_time_hours
- resolution_time_hours
- created_at

5.6 Relations et cardinalités (Schéma 2)

Tenant (1) — (0..) Invoice*

- Un tenant peut générer plusieurs factures.

Contact (1) — (0..) Invoice*

- Une facture est rattachée à un contact client (customer_contact_id).

Invoice (1) — (0..) Payment*

- Une facture peut être payée en un ou plusieurs paiements (ex : paiement partiel).

Tenant (1) — (0..) PaymentMethod*

- Un tenant définit ses modes de paiement.

User (1) — (0..) PhoneCall*

- Un user peut passer/recevoir plusieurs appels.

Contact (0..1) — (0..) PhoneCall*

- Un appel peut être lié à un contact (contact_id nullable).

User (1) — (0..) SupportTicket (created_by)*

- Un user peut créer plusieurs tickets.

User (0..1) — (0..) SupportTicket (assigned_to)*

- Un ticket peut être assigné à un user (nullable).

Tenant (1) — (0..) SLA*

- Un tenant peut avoir un ou plusieurs SLA (selon l'offre/pack).

Note : on peut ajouter plus tard une FK SupportTicket.sla_id si on souhaite lier chaque ticket à un SLA précis (option d'évolution).

5.7 Contraintes et règles transverses (Schémas 1 + 2)

- **Intégrité multi-tenant :** tenant_id obligatoire partout (sauf tables de liaison internes) afin de garantir l'isolation.
- **Suppression logique recommandée** (soft delete) pour les données CRM sensibles (Contact/Lead/Opportunity) afin de respecter audit + RGPD.
- **Index recommandés :**
    - (tenant_id) sur toutes les entités
    - (tenant_id, email) sur User et Contact
    - (tenant_id, status) sur Lead, Opportunity, Ticket
    - (opportunity_id) sur Activity si requêtes fréquentes par opportunité
- **Backups :** sauvegardes régulières + stratégie de restauration (cf. livrable infrastructure).

5.1 Principe Multi-Tenant (commun à tout le modèle)

La plateforme CRM est **multi-tenant** : plusieurs entreprises clientes (tenants) partagent la même application, mais leurs données doivent rester **strictement isolées**.
Chaque entité métier contient donc un champ tenant_id (FK) qui rattache l'enregistrement à une entreprise donnée.

Règle d'isolation :

toute requête (lecture/écriture) doit être filtrée par tenant_id afin d'empêcher l'accès aux données d'un autre tenant.

5.4 Objectif du schéma 2

Le schéma 2 présente des entités **volontairement hors périmètre** (facturation, paiement, téléphonie, support/SLA), afin de démontrer l'évolutivité de l'architecture et la possibilité d'étendre le CRM sans refonte du cœur métier.

5.5 Classes / Entités ajoutées (Schéma 2)

A) Invoice (Facturation)

Rôle : gestion des factures d'abonnement (hors périmètre officiel).
Attributs :

- invoice_id (PK)

- tenant_id (FK -> Tenant.tenant_id)
- customer_contact_id (FK -> Contact.contact_id)
- invoice_number
- amount_total
- currency
- status (DRAFT / SENT / PAID / CANCELLED)
- issued_at
- due_date
- created_at

B) Payment (Paiement)

Rôle : enregistrement des paiements liés à une facture.

Attributs :

- payment_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- invoice_id (FK -> Invoice.invoice_id)
- amount
- payment_date
- status (PENDING / SUCCESS / FAILED)
- created_at

C) PaymentMethod (Mode de paiement)

Rôle : mode de paiement configuré par tenant (carte, virement, PayPal…).

Attributs :

- payment_method_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- type (CARD / BANK_TRANSFER / PAYPAL)
- provider
- is_default
- created_at

D) PhoneCall (Téléphonie intégrée)

Rôle : gestion avancée des appels (si téléphonie intégrée).

Attributs :

- call_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- contact_id (FK -> Contact.contact_id, nullable)
- user_id (FK -> User.user_id)
- direction (IN / OUT)
- started_at
- duration_seconds
- status (COMPLETED / MISSED)
- created_at

E) SupportTicket (Support client)

Rôle : tickets de support client (hors périmètre officiel).

Attributs :

- ticket_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- created_by_user_id (FK -> User.user_id)
- assigned_user_id (FK -> User.user_id, nullable)
- subject
- status (OPEN / IN_PROGRESS / CLOSED)
- priority (LOW / MEDIUM / HIGH)
- created_at
- closed_at (nullable)

F) SLA (Service Level Agreement)

Rôle : définition des engagements de support par tenant (temps de réponse / résolution).

Attributs :

- sla_id (PK)
- tenant_id (FK -> Tenant.tenant_id)
- name

- response_time_hours
- resolution_time_hours
- created_at

5.6 Relations et cardinalités (Schéma 2)

Tenant (1) — (0..) Invoice*

- Un tenant peut générer plusieurs factures.

Contact (1) — (0..) Invoice*

- Une facture est rattachée à un contact client (customer_contact_id).

Invoice (1) — (0..) Payment*

- Une facture peut être payée en un ou plusieurs paiements (ex : paiement partiel).

Tenant (1) — (0..) PaymentMethod*

- Un tenant définit ses modes de paiement.

User (1) — (0..) PhoneCall*

- Un user peut passer/recevoir plusieurs appels.

Contact (0..1) — (0..) PhoneCall*

- Un appel peut être lié à un contact (contact_id nullable).

User (1) — (0..) SupportTicket (created_by)*

- Un user peut créer plusieurs tickets.

User (0..1) — (0..) SupportTicket (assigned_to)*

- Un ticket peut être assigné à un user (nullable).

Tenant (1) — (0..) SLA*

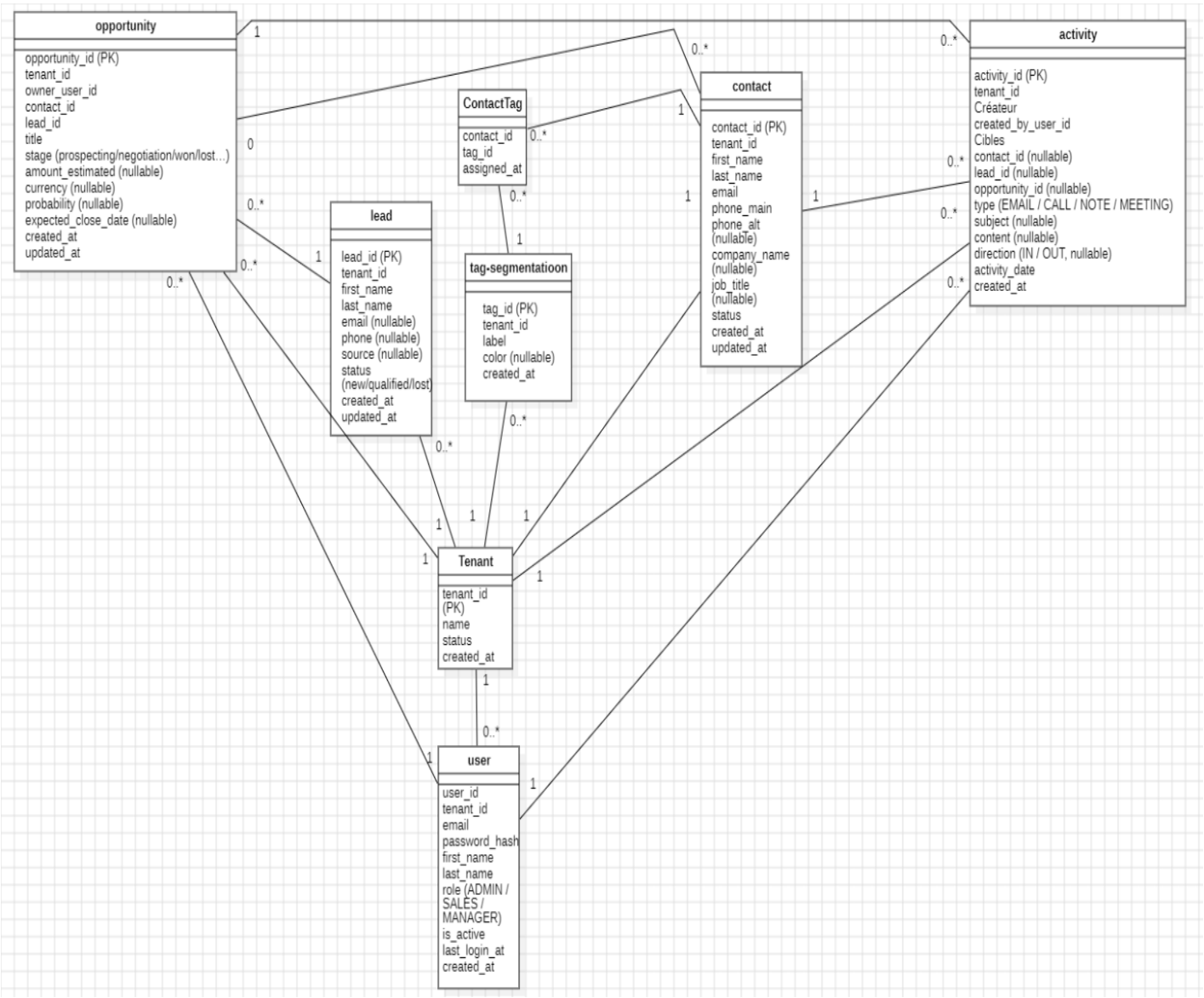
- Un tenant peut avoir un ou plusieurs SLA (selon l'offre/pack).

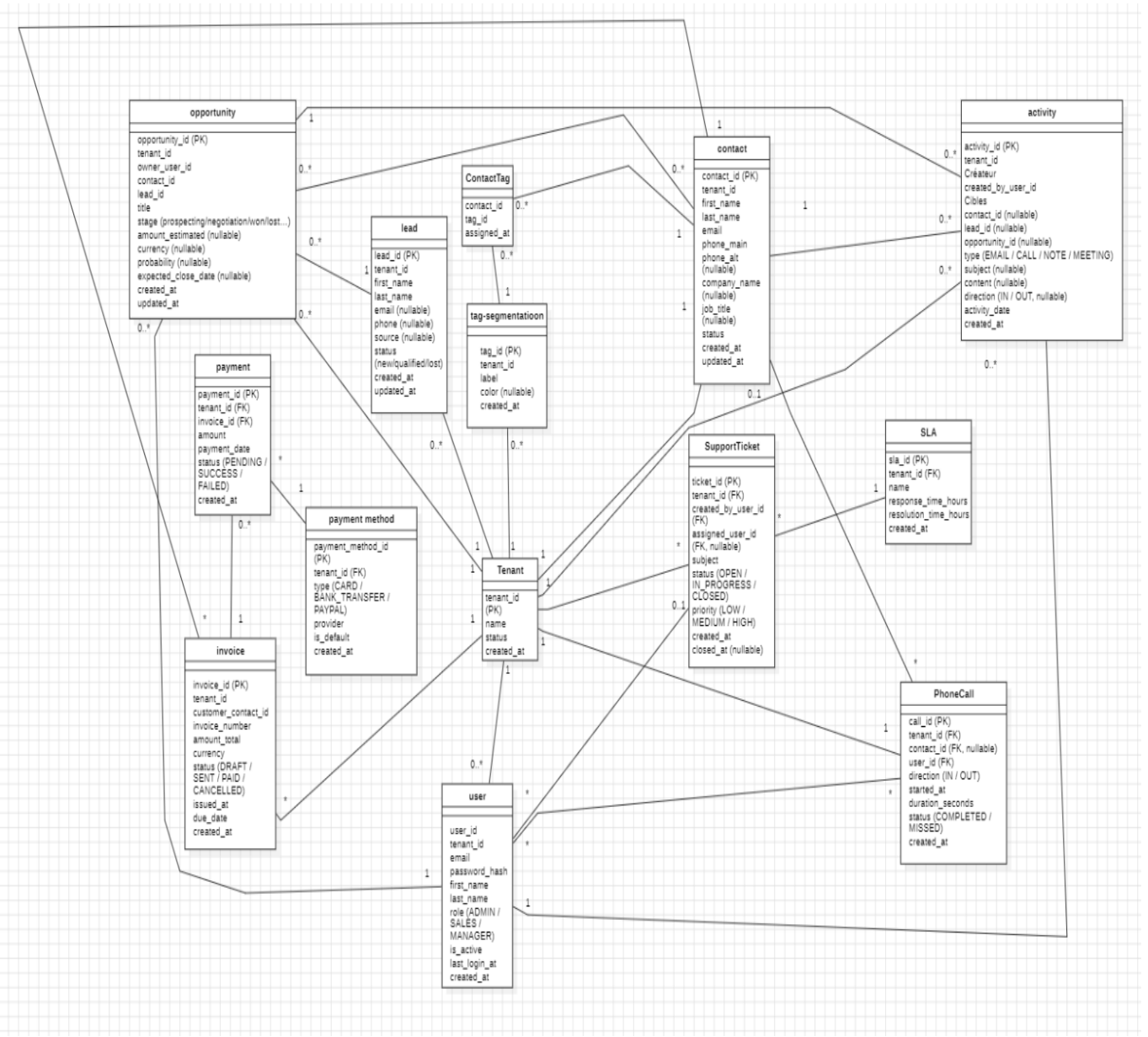
Note : on peut ajouter plus tard une FK SupportTicket.sla_id si on souhaite lier chaque ticket à un SLA précis (option d'évolution).

5.7 Contraintes et règles transverses (Schémas 1 + 2)

- **Intégrité multi-tenant :** tenant_id obligatoire partout (sauf tables de liaison internes) afin de garantir l'isolation.
- **Suppression logique recommandée** (soft delete) pour les données CRM sensibles (Contact/Lead/Opportunity) afin de respecter audit + RGPD.
- **Index recommandés :**
  - (tenant_id) sur toutes les entités

    - (tenant_id, email) sur User et Contact
    - (tenant_id, status) sur Lead, Opportunity, Ticket
    - (opportunity_id) sur Activity si requêtes fréquentes par opportunité
- **Backups :** sauvegardes régulières + stratégie de restauration (cf. livrable infrastructure).


opportunity
opportunity_id (PK)
tenant_id
owner_user_id
contact_id
lead_id
title
stage (prospecting/negotiation/won/lost…)
amount_estimated (nullable)
currency (nullable)
probability (nullable)
expected_close_date (nullable)
created_at
updated_at
ContactTag
contact_id
tag_id
assigned_at
contact
contact_id (PK)
tenant_id
first_name
last_name
email
phone_main
phone_alt (nullable)
company_name (nullable)
job_title (nullable)
status
created_at
updated_at
activity
activity_id (PK)
tenant_id
Créateur
created_by_user_id
Cibles
contact_id (nullable)
lead_id (nullable)
opportunity_id (nullable)
type (EMAIL / CALL / NOTE / MEETING)
subject (nullable)
content (nullable)
direction (IN / OUT, nullable)
activity_date
created_at
lead
lead_id (PK)
tenant_id
first_name
last_name
email (nullable)
phone (nullable)
source (nullable)
status (new/qualified/lost)
created_at
updated_at
tag-segmentatioon
tag_id (PK)
tenant_id
label
color (nullable)
created_at
Tenant
tenant_id (PK)
name
status
created_at
user
user_id
tenant_id
email
password_hash
first_name
last_name
role (ADMIN / SALES / MANAGER)
is_active
last_login_at
created_at



ADR

ADR 01 — Choix d'une architecture SaaS multi-tenant

Contexte

La plateforme CRM est conçue pour être utilisée par plusieurs entreprises clientes via Internet.

Problème
Comment concevoir une architecture permettant à plusieurs entreprises d'utiliser la même application tout en garantissant l'isolation des données ?

Options envisagées

1. Une application dédiée par client (single-tenant)
2. Une application partagée multi-tenant avec isolation logique des données

Critères de décision

- Scalabilité
- Coûts d'exploitation
- Facilité de maintenance
- Alignement avec le modèle SaaS

Décision retenue
Adoption d'une **architecture multi-tenant avec isolation logique via tenant_id**.

Conséquences / risques

- Réduction des coûts et meilleure scalabilité
- Nécessité d'une vigilance stricte sur le filtrage par tenant_id
- Risque de fuite de données en cas de mauvaise implémentation applicative

ADR 02 — Choix d'une infrastructure Cloud

Contexte
La plateforme CRM doit être accessible en ligne et capable de monter en charge rapidement.

Problème
Faut-il héberger la solution en on-premise, cloud ou hybride ?

Options envisagées

1. Infrastructure on-premise
2. Infrastructure cloud public
3. Architecture hybride

Critères de décision

- Scalabilité
- Disponibilité

- Rapidité de déploiement
- Coût et flexibilité

Décision retenue
Choix d'une **infrastructure Cloud**.

Conséquences / risques

- Excellente élasticité et haute disponibilité
- Dépendance au fournisseur cloud (vendor lock-in)
- Coûts variables à surveiller

ADR 03 — Découpage en zones réseau (DMZ / Applicative / Données)

Contexte
La plateforme est exposée sur Internet et traite des données sensibles.

Problème
Comment sécuriser l'accès réseau tout en respectant les bonnes pratiques d'architecture ?

Options envisagées

1. Réseau plat sans segmentation
2. Segmentation en zones de sécurité

Critères de décision

- Sécurité
- Principe du moindre privilège
- Lisibilité de l'architecture

Décision retenue
Adoption d'une **architecture réseau segmentée** :
Internet → DMZ → Zone Applicative → Zone Données.

Conséquences / risques

- Sécurité renforcée
- Complexité de configuration réseau accrue
- Besoin de documentation claire des flux autorisés

ADR 04 — Utilisation d'un Load Balancer en frontal

Contexte
La plateforme doit supporter plusieurs utilisateurs simultanés.

Problème
Comment répartir la charge et éviter les points de défaillance uniques ?

Options envisagées

1. Accès direct aux serveurs applicatifs
2. Load balancer HTTPS en frontal

Critères de décision

- Disponibilité
- Scalabilité
- Tolérance aux pannes

Décision retenue
Mise en place d'un **load balancer HTTPS** en zone publique.

Conséquences / risques

- Répartition automatique de la charge
- Point critique nécessitant redondance
- Configuration TLS à maintenir

ADR 05 — Choix d'une API REST pour la communication applicative

Contexte
Le frontend et les services backend doivent communiquer efficacement.

Problème
Quel style d'API adopter ?

Options envisagées

1. REST
2. GraphQL
3. gRPC

Critères de décision

- Simplicité

- Compatibilité avec les clients web
- Lisibilité et maturité

Décision retenue
Adoption d'une **API REST sur HTTPS**.

Conséquences / risques

- Facilité d'intégration
- Risque d'over-fetching
- Possibilité d'évolution future vers GraphQL

ADR 06 — Authentification via OAuth2 / OIDC avec JWT

Contexte
La plateforme nécessite une authentification sécurisée et scalable.

Problème
Comment gérer l'authentification sans sessions serveur ?

Options envisagées

1. Sessions serveur classiques
2. JWT avec OAuth2 / OpenID Connect

Critères de décision

- Scalabilité
- Sécurité
- Architecture stateless

Décision retenue
Utilisation de **JWT avec OAuth2 / OIDC**.

Conséquences / risques

- Architecture stateless et scalable
- Gestion délicate de la révocation des tokens
- Nécessité de durées de vie courtes des tokens

ADR 07 — Choix d’une base de données relationnelle SQL

Contexte
Le CRM gère des données fortement structurées et transactionnelles.

Problème
Quel type de base de données utiliser ?

Options envisagées

1. Base relationnelle SQL
2. Base NoSQL

Critères de décision

- Cohérence des données
- Relations complexes
- Transactions ACID

Décision retenue
Adoption d’une **base de données relationnelle SQL**.

Conséquences / risques

- Modélisation claire et robuste
- Scalabilité horizontale plus complexe
- Nécessité d’optimisation (index, cache)

ADR 08 — Utilisation de Redis comme cache

Contexte
Certaines données CRM sont fréquemment consultées.

Problème
Comment réduire la charge sur la base de données ?

Options envisagées

1. Accès direct à la base de données
2. Cache en mémoire (Redis)

Critères de décision

- Performance
- Réduction de la latence

- Simplicité d’intégration

Décision retenue
Utilisation de **Redis comme cache applicatif**.

Conséquences / risques

- Amélioration des performances
- Gestion de l’invalidation du cache
- Dépendance à un service supplémentaire

ADR 09 — Conteneurisation avec Docker

Contexte
Le déploiement doit être reproductible et portable.

Problème
Comment standardiser les environnements de déploiement ?

Options envisagées

1. Déploiement natif sur VM
2. Conteneurisation Docker

Critères de décision

- Portabilité
- Reproductibilité
- Alignement avec les pratiques modernes

Décision retenue
Utilisation de **Docker pour les services applicatifs**.

Conséquences / risques

- Déploiement simplifié
- Besoin de gestion des images
- Complexité supplémentaire pour l’orchestration

ADR 10 — Centralisation des logs et monitoring

Contexte
La plateforme doit être surveillée et auditée.

Problème
Comment diagnostiquer efficacement les incidents ?

Options envisagées

1. Logs locaux non centralisés
2. Centralisation des logs et monitoring

Critères de décision

- Observabilité
- Diagnostic des incidents
- Sécurité et audit

Décision retenue
Centralisation des logs et métriques via des outils dédiés.

Conséquences / risques

- Meilleure visibilité sur le système
- Coût de stockage des logs
- Nécessité de définir des alertes pertinentes

7. Analyse des Risques

L'analyse des risques vise à identifier les menaces potentielles pouvant affecter la performance, la sécurité, la disponibilité et la pérennité de la plateforme CRM SaaS.
Elle permet d'anticiper les problèmes, d'évaluer leur gravité et de définir des actions de mitigation adaptées.

Cette analyse couvre les **risques techniques, de sécurité, de performance, de scalabilité ainsi que les risques liés aux coûts et à la dette technique**, conformément aux bonnes pratiques d'architecture étudiées.

7.1 Risques Techniques

Risque : Point de défaillance unique (SPOF)

- **Description** : Dépendance à un composant critique (base de données, load balancer).
- **Probabilité** : Moyenne
- **Impact** : Critique
- **Détection** : Indisponibilité soudaine du service, alertes monitoring
- **Mitigation** :
    - Load balancer redondant
    - Réplication de la base de données
    - Tests de bascule réguliers (failover)

Risque : Complexité excessive de l'architecture

- **Description** : Accumulation de composants techniques non nécessaires.
- **Probabilité** : Faible à moyenne
- **Impact** : Modéré
- **Détection** : Difficultés de maintenance, bugs récurrents
- **Mitigation** :
    - Principe KISS (Keep It Simple)
    - Documentation claire (DAT, ADR)
    - Revues d'architecture régulières

7.2 Risques Sécurité

Risque : Fuite de données entre tenants

- **Description** : Mauvaise gestion du tenant_id entraînant un accès inter-entreprises.
- **Probabilité** : Faible
- **Impact** : Critique
- **Détection** : Audits de sécurité, logs d'accès, tests automatisés
- **Mitigation** :
    - Filtrage systématique par tenant_id
    - Tests unitaires et d'intégration multi-tenant
    - Principe du moindre privilège

Risque : Attaques applicatives (SQL injection, XSS, CSRF)

- **Probabilité** : Moyenne
- **Impact** : Élevé
- **Détection** : Logs de sécurité, scans OWASP
- **Mitigation** :
    - Validation stricte des entrées
    - ORM et requêtes préparées
    - HTTPS obligatoire
    - Tests de sécurité réguliers

7.3 Risques de Performance

Risque : Dégradation des performances sous charge

- **Description** : Requêtes lourdes, absence d'index ou de cache.
- **Probabilité** : Moyenne
- **Impact** : Élevé
- **Détection** : Augmentation de la latence, alertes métriques
- **Mitigation** :
    - Mise en cache Redis
    - Indexation des tables critiques
    - Monitoring des temps de réponse

7.4 Risques de Scalabilité

Risque : Scalabilité insuffisante face à la croissance

- **Description** : Incapacité à absorber une augmentation rapide des utilisateurs.
- **Probabilité** : Moyenne
- **Impact** : Critique
- **Détection** : Saturation CPU/RAM, erreurs applicatives
- **Mitigation** :
    - Architecture stateless
    - Scaling horizontal
    - Auto-scaling basé sur métriques

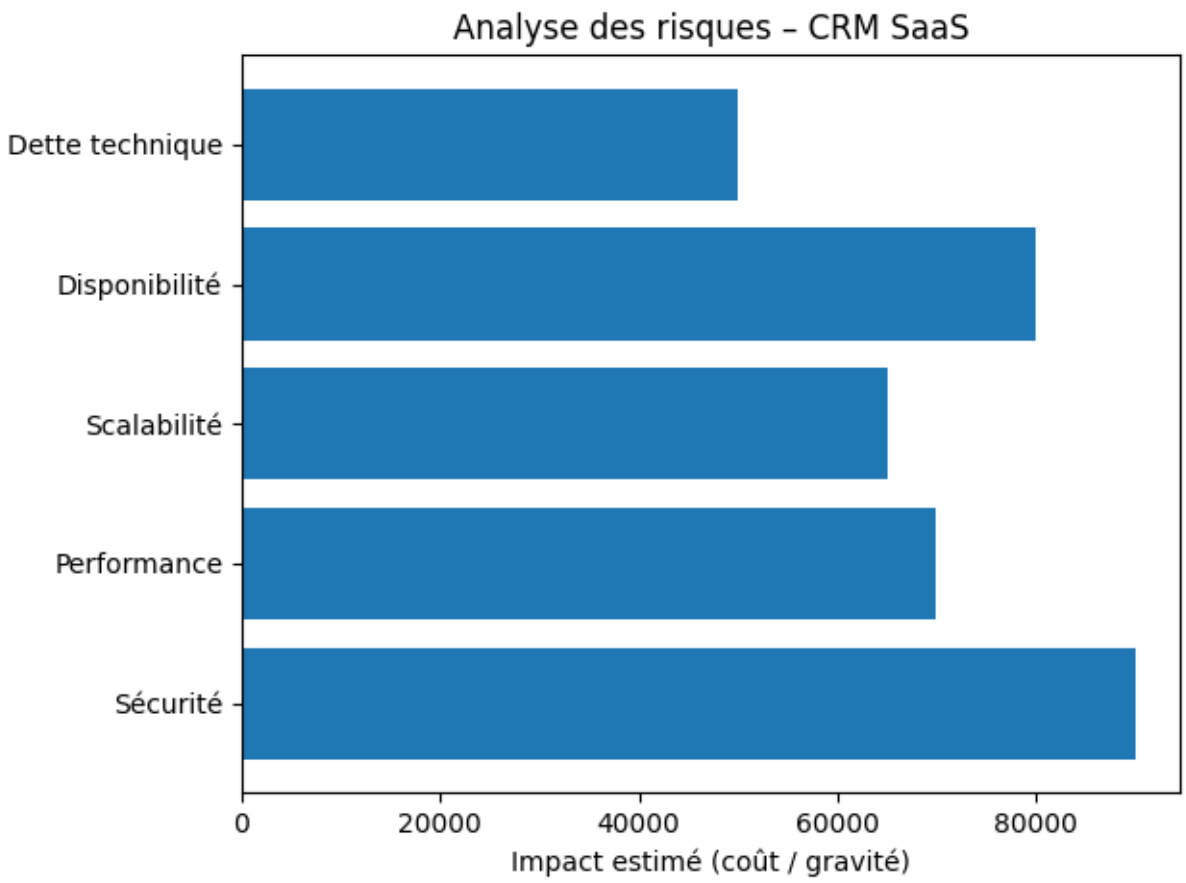
7.5 Risques Coûts & Dette Technique

Risque : Explosion des coûts d'infrastructure cloud

- **Description** : Mauvaise maîtrise du scaling automatique.
- **Probabilité** : Moyenne
- **Impact** : Modéré à élevé
- **Détection** : Suivi budgétaire cloud, alertes de coûts
- **Mitigation** :
    - Seuils d'auto-scaling contrôlés
    - Monitoring des coûts
    - Optimisation des ressources

Risque : Accumulation de dette technique

- **Description** : Choix rapides non documentés, absence de refactoring.
- **Probabilité** : Moyenne
- **Impact** : Modéré
- **Détection** : Difficulté d'évolution, bugs fréquents
- **Mitigation** :
    - Utilisation des ADR
    - Refactoring régulier
    - Standards de développemen



Sécurité représente le risque le plus critique, en raison de la nature sensible des données manipulées (données clients, informations commerciales, données personnelles soumises au RGPD).

Disponibilité constitue un risque majeur dans un contexte SaaS, où une indisponibilité impacte directement l'ensemble des entreprises clientes.

Performance et **Scalabilité** sont des risques importants liés à la croissance du nombre de tenants et d'utilisateurs simultanés.

Dette technique présente un impact plus modéré à court terme, mais peut devenir critique sur le long terme si l'architecture n'est pas maîtrisée.