Manuel Développeur - CRM Freelance

Version 1.0 - Décembre 2025

Table des matières

Architecture technique

Stack technologique

Catégorie	Technologies
Frontend	Next.js 15 (App Router), React 18, TypeScript
Styling	Tailwind CSS, CSS Variables (thème clair/sombre)
Backend	Next.js API Routes (route handlers), Prisma ORM
Auth	Auth.js v5 (ex-NextAuth), bcryptjs
Base de données	PostgreSQL
State Management	React Query (TanStack Query)
PDF	@react-pdf/renderer
Paiements	Stripe
Déploiement	Vercel

Principes architecturaux

- **App Router** : Utilisation du nouveau routeur Next.js 15
- **Server Components** : Par défaut, composants côté serveur
- **API Routes** : Route handlers pour les endpoints REST
- **Prisma ORM** : Accès type-safe à la base de données
- **React Query** : Cache et synchronisation des données

Installation et configuration

Prérequis

- Node.js 18+
- PostgreSQL 14+
- npm ou yarn

Installation

```
# Cloner le repository
git clone https://github.com/votre-repo/crm-freelance.git
cd crm-freelance

# Installer les dépendances
npm install

# Copier les variables d'environnement
cp .env.example .env
```

Variables d'environnement

```
# Base de données PostgreSQL
DATABASE_URL="postgresql://user:password@localhost:5432/crm_freelance?
schema=public"

# NextAuth (obligatoire)
NEXTAUTH_SECRET="une-chaine-secrete-aleatoire-longue-32-caracteres"
NEXTAUTH_URL="http://localhost:3000"

# Stripe (optionnel)
STRIPE_SECRET_KEY="sk_test_..."
STRIPE_WEBHOOK_SECRET="whsec_..."
```

Initialisation de la base de données

```
# Générer le client Prisma
npm run db:generate

# Appliquer les migrations
npm run db:push

# (Optionnel) Ouvrir Prisma Studio
npm run db:studio
```

Lancer le serveur de développement

```
npm run dev
```

L'application est accessible sur `http://localhost:3000`.

Structure du projet

```
crm/
├── app/                          # Pages Next.js (App Router)
│   ├── (dashboard)/              # Routes authentifiées avec sidebar
│   │   ├── clients/              # Liste + fiche client [id]
│   │   ├── opportunites/         # Pipeline Kanban
│   │   ├── parametres/           # Paramètres utilisateur
│   │   └── tickets/              # Liste + fiche ticket [id]
│   ├── api/                      # Routes API
│   │   ├── auth/                 # NextAuth + inscription
│   │   ├── clients/              # CRUD clients
│   │   ├── contacts/             # CRUD contacts
│   │   ├── documents/            # CRUD documents
│   │   ├── opportunites/         # CRUD opportunités + PDF
│   │   ├── paiements/            # Sessions Stripe
│   │   ├── portail/              # API portail client
│   │   ├── tickets/              # CRUD tickets
│   │   ├── upload/               # Upload fichiers
│   │   ├── utilisateur/          # Infos entreprise
│   │   └── webhooks/             # Webhooks Stripe
│   ├── auth/                     # Pages authentification
│   └── portail/                  # Portail client (public)
├── components/                   # Composants React
│   ├── clients/                  # Modales client
│   ├── contacts/                 # Modales contact
│   ├── emails/                   # Modales email
│   ├── filtres/                  # Panneau filtres
│   ├── layout/                   # Sidebar, PageHeader
│   ├── opportunites/             # Kanban, modales, documents
│   ├── providers/                # QueryProvider, SessionProvider
│   ├── theme/                    # ThemeProvider, ThemeToggle
│   ├── tickets/                  # Modales ticket
│   └── ui/                       # Toast, Recherche, Export
├── lib/                          # Utilitaires
│   ├── api.ts                    # Fonctions fetch API
│   ├── auth.ts                   # Config NextAuth
│   ├── hooks.ts                  # Hooks React Query
│   ├── integrations/             # Intégrations tierces
│   │   └── stripe.ts             # Configuration Stripe
│   ├── pdf/                      # Génération PDF
│   │   └── DocumentPDF.tsx       # Composant React-PDF
│   ├── portail.ts                # Utilitaires portail
```

```
│   ├── prisma.ts                # Client Prisma singleton
│   ├── utils.ts                 # Helpers (dates, montants)
│   └── validateurs.ts           # Schémas Zod
├── prisma/
│   ├── migrations/              # Migrations SQL
│   └── schema.prisma            # Modèle de données
├── public/
│   └── uploads/                 # Fichiers uploadés
├── types/
│   ├── dto.ts                   # Interfaces DTO
│   ├── models.ts                # Interfaces modèles
│   └── next-auth.d.ts           # Types NextAuth étendus
└── middleware.ts                # Protection des routes
```

Base de données

Modèle de données (Prisma)

```
model User {
  id                String    @id @default(cuid())
  email             String    @unique
  motDePasse        String?
  nomAffiche        String?
  logoUrl           String?

  // Informations entreprise
  raisonSociale     String?
  adresseLigne1     String?
  adresseLigne2     String?
  codePostal        String?
  ville             String?
  pays              String?
  siret             String?
  numeroTva         String?
  telephone         String?
  iban              String?
  bic               String?
  mentionsLegales   String?
  tauxTva           Float     @default(0)

  // Compteurs documents
  compteurDevis     Int       @default(0)
  compteurFacture   Int       @default(0)

  // Relations
  clients           Client[]
  opportunites      Opportunite[]
  tickets           Ticket[]
}
```

```
model Client {
  id                String    @id @default(cuid())
  nom               String
  emailPrincipal    String?
  telephone         String?
  statut            String    @default("prospect")
  type              String    @default("particulier")

  // Informations entreprise
  raisonSociale     String?
  siteWeb           String?
  adresseLigne1     String?
  adresseLigne2     String?
  codePostal        String?
  ville             String?
  pays              String?
  siret             String?
  numeroTva         String?
  secteurActivite   String?
  tailleEntreprise  String?

  // Relations
  proprietaireId    String
  proprietaire      User      @relation(fields: [proprietaireId])
  contacts          Contact[]
  opportunites      Opportunite[]
  tickets           Ticket[]
}

model Opportunite {
  id                String    @id @default(cuid())
  titre             String
  descriptionCourte String?
  montantEstime     Float?
  devise            String    @default("EUR")
  etapePipeline     String    @default("lead")
  statutPaiement    String    @default("en_attente")

  // Relations
  clientId          String
  client            Client    @relation(fields: [clientId])
  proprietaireId    String
  proprietaire      User      @relation(fields: [proprietaireId])
  documents         Document[]
}

model Document {
  id                String    @id @default(cuid())
  nom               String
  typeDocument      String    @default("autre")
  fichierUrl        String
  visiblePortail    Boolean   @default(false)

  opportuniteId     String
```

```
  opportunite       Opportunite @relation(fields: [opportuniteId])
}

model Ticket {
  id                String    @id @default(cuid())
  titre             String
  description       String?
  statut            String    @default("ouvert")
  priorite          String    @default("normale")
  type              String    @default("autre")

  clientId          String
  client            Client    @relation(fields: [clientId])
  assigneId         String?
  assigne           User?     @relation(fields: [assigneId])
}
```

Migrations

```
# Créer une nouvelle migration
npx prisma migrate dev --name nom_migration

# Appliquer les migrations en production
npx prisma migrate deploy

# Réinitialiser la base (dev uniquement)
npx prisma migrate reset
```

Authentification

Configuration NextAuth

```
// lib/auth.ts
import NextAuth from 'next-auth';
import Credentials from 'next-auth/providers/credentials';
import bcrypt from 'bcryptjs';
import { prisma } from './prisma';

export const { handlers, auth, signIn, signOut } = NextAuth({
  providers: [
    Credentials({
      credentials: {
        email: { label: 'Email', type: 'email' },
        password: { label: 'Mot de passe', type: 'password' },
      },
      async authorize(credentials) {
        const user = await prisma.user.findUnique({
          where: { email: credentials.email },
```

```
        });

        if (!user || !user.motDePasse) return null;

        const valid = await bcrypt.compare(
          credentials.password,
          user.motDePasse
        );

        if (!valid) return null;

        return { id: user.id, email: user.email, name: user.nomAffiche };
      },
    }),
  ],
  callbacks: {
    jwt({ token, user }) {
      if (user) token.id = user.id;
      return token;
    },
    session({ session, token }) {
      session.user.id = token.id;
      return session;
    },
  },
});
```

Protection des routes

```
// middleware.ts
import { auth } from '@/lib/auth';

export default auth((req) => {
  const isLoggedIn = !!req.auth;
  const isAuthPage = req.nextUrl.pathname.startsWith('/auth');
  const isApiAuth = req.nextUrl.pathname.startsWith('/api/auth');
  const isPortail = req.nextUrl.pathname.startsWith('/portail');

  if (isApiAuth || isPortail) return;

  if (!isLoggedIn && !isAuthPage) {
    return Response.redirect(new URL('/auth/connexion', req.nextUrl));
  }

  if (isLoggedIn && isAuthPage) {
    return Response.redirect(new URL('/', req.nextUrl));
  }
});

export const config = {
```

```
  matcher: ['/((?!_next/static|_next/image|favicon.ico|uploads).*)'],
};
```

Récupérer la session

```
// Dans un Server Component
import { auth } from '@/lib/auth';

export default async function Page() {
  const session = await auth();
  if (!session) redirect('/auth/connexion');

  return <div>Bonjour {session.user.name}</div>;
}

// Dans une API Route
import { auth } from '@/lib/auth';

export async function GET() {
  const session = await auth();
  if (!session) {
    return Response.json({ error: 'Non autorisé' }, { status: 401 });
  }
  // ...
}
```

API REST

Convention des routes

Méthode	Route	Description
GET	`/api/clients`	Liste des clients
POST	`/api/clients`	Créer un client
GET	`/api/clients/[id]`	Détail d'un client
PATCH	`/api/clients/[id]`	Modifier un client
DELETE	`/api/clients/[id]`	Supprimer un client

Exemple de route API

```
// app/api/clients/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { auth } from '@/lib/auth';
import { prisma } from '@/lib/prisma';
import { z } from 'zod';
```

```
const clientSchema = z.object({
  nom: z.string().min(1),
  emailPrincipal: z.string().email().optional(),
  telephone: z.string().optional(),
  statut: z.enum(['prospect', 'actif', 'inactif']).optional(),
});

export async function GET(request: NextRequest) {
  const session = await auth();
  if (!session) {
    return NextResponse.json({ error: 'Non autorisé' }, { status: 401 });
  }

  const clients = await prisma.client.findMany({
    where: { proprietaireId: session.user.id },
    orderBy: { dateCreation: 'desc' },
  });

  return NextResponse.json(clients);
}

export async function POST(request: NextRequest) {
  const session = await auth();
  if (!session) {
    return NextResponse.json({ error: 'Non autorisé' }, { status: 401 });
  }

  const body = await request.json();
  const validation = clientSchema.safeParse(body);

  if (!validation.success) {
    return NextResponse.json(
      { error: 'Données invalides', details: validation.error.errors },
      { status: 400 }
    );
  }

  const client = await prisma.client.create({
    data: {
      ...validation.data,
      proprietaireId: session.user.id,
    },
  });

  return NextResponse.json(client, { status: 201 });
}
```

API Utilisateur (Paramètres)

```
// GET /api/utilisateur
// Récupère les informations de l'utilisateur connecté

// PATCH /api/utilisateur
// Met à jour les informations entreprise
{
  "raisonSociale": "Ma Société SAS",
  "adresseLigne1": "123 rue Example",
  "codePostal": "75001",
  "ville": "Paris",
  "siret": "12345678901234",
  "tauxTva": 20
}
```

API Génération PDF

```
// GET /api/opportunites/[id]/pdf?type=devis
// Génère un devis PDF

// GET /api/opportunites/[id]/pdf?type=facture
// Génère une facture PDF (si opportunité gagnée ET payée)
```

Composants React

Hooks personnalisés (React Query)

```
// lib/hooks.ts
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';

export function useClients() {
  return useQuery({
    queryKey: ['clients'],
    queryFn: async () => {
      const res = await fetch('/api/clients');
      if (!res.ok) throw new Error('Erreur');
      return res.json();
    },
  });
}

export function useCreerClient() {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: async (data: ClientInput) => {
      const res = await fetch('/api/clients', {
        method: 'POST',
```

```
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(data),
      });
      if (!res.ok) throw new Error('Erreur');
      return res.json();
    },
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['clients'] });
    },
  });
}
```

Composant Toast

```
// Utilisation
import { useToast } from '@/components/ui/Toast';

function MonComposant() {
  const toast = useToast();

  const handleClick = () => {
    toast.success('Opération réussie');
    toast.error('Une erreur est survenue');
    toast.info('Information');
  };
}
```

Kanban Board

```
// components/opportunites/KanbanBoard.tsx
// Utilise @dnd-kit pour le drag & drop

import {
  DndContext,
  DragOverlay,
  closestCorners,
} from '@dnd-kit/core';

// Les colonnes sont définies par étape de pipeline
const ETAPES = ['lead', 'qualification', 'proposition', 'negociation', 'gagne',
'perdu'];
```

Génération PDF

Composant DocumentPDF

```tsx
// lib/pdf/DocumentPDF.tsx
import { Document, Page, Text, View, Image, StyleSheet } from '@react-
pdf/renderer';

interface DocumentPDFProps {
  typeDocument: 'devis' | 'facture';
  numeroDocument: string;
  dateEmission: Date;
  emetteur: DonneesEmetteur;
  destinataire: DonneesDestinataire;
  opportunite: DonneesOpportunite;
  estPaye?: boolean;
}

export function DocumentPDF({
  typeDocument,
  numeroDocument,
  emetteur,
  destinataire,
  opportunite,
  estPaye,
}: DocumentPDFProps) {
  return (
    <Document>
      <Page size="A4" style={styles.page}>
        {/* Tampon PAYÉ */}
        {typeDocument === 'facture' && estPaye && (
          <View style={styles.tamponPaye}>
            <Text style={styles.tamponPayeTexte}>PAYÉ</Text>
          </View>
        )}

        {/* En-tête avec logo */}
        <View style={styles.header}>
          {emetteur.logoUrl && (
            <Image src={emetteur.logoUrl} style={styles.logo} />
          )}
          {/* ... */}
        </View>

        {/* Total avec TVA */}
        {emetteur.tauxTva > 0 ? (
          <View>
            <Text>Total HT : {montantHT} €</Text>
            <Text>TVA ({emetteur.tauxTva}%) : {montantTVA} €</Text>
            <Text>Total TTC : {montantTTC} €</Text>
          </View>
        ) : (
          <View>
            <Text>Total : {montant} €</Text>
            <Text>TVA non applicable, art. 293 B du CGI</Text>
          </View>
        )}
```

```
      </Page>
    </Document>
  );
}
```

Route API de génération

```
// app/api/opportunites/[id]/pdf/route.ts
import { renderToBuffer } from '@react-pdf/renderer';
import { DocumentPDF } from '@/lib/pdf/DocumentPDF';

export async function GET(request: NextRequest, { params }) {
  const { id } = params;
  const type = request.nextUrl.searchParams.get('type');

  // Vérifier les conditions pour la facture
  if (type === 'facture') {
    if (opportunite.etapePipeline !== 'gagne' ||
        opportunite.statutPaiement !== 'paye') {
      return NextResponse.json(
        { error: 'Facture non disponible' },
        { status: 400 }
      );
    }
  }

  // Incrémenter le compteur
  const compteur = type === 'devis' ? 'compteurDevis' : 'compteurFacture';
  await prisma.user.update({
    where: { id: session.user.id },
    data: { [compteur]: { increment: 1 } },
  });

  // Générer le numéro
  const annee = new Date().getFullYear();
  const prefixe = type === 'devis' ? 'DEV' : 'FAC';
  const numero = `${prefixe}-${annee}-${String(compteur).padStart(4, '0')}`;

  // Convertir le logo en base64
  let logoBase64 = null;
  if (user.logoUrl) {
    const logoPath = path.join(process.cwd(), 'public', user.logoUrl);
    const logoBuffer = await readFile(logoPath);
    const ext = path.extname(logoPath).slice(1);
    logoBase64 = `${logoBuffer.toString('base64')}`;
  }

  // Générer le PDF
  const pdfBuffer = await renderToBuffer(
    <DocumentPDF
      typeDocument={type}
```

```
        numeroDocument={numero}
        emetteur={{ ...user, logoUrl: logoBase64 }}
        destinataire={client}
        opportunite={opportunite}
        estPaye={opportunite.statutPaiement === 'paye'}
      />
    );

    // Sauvegarder dans les documents
    await prisma.document.create({
      data: {
        opportuniteId: id,
        nom: `${type}_${numero}.pdf`,
        typeDocument: type,
        fichierUrl: `/uploads/${session.user.id}/${filename}`,
      },
    });

    return new Response(pdfBuffer, {
      headers: {
        'Content-Type': 'application/pdf',
        'Content-Disposition': `inline; filename="${filename}"`,
      },
    });
  }
```

Intégrations tierces

Stripe

```
// lib/integrations/stripe.ts
import Stripe from 'stripe';

export const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2023-10-16',
});

// Créer une session de paiement
export async function creerSessionPaiement(opportunite: Opportunite) {
  const session = await stripe.checkout.sessions.create({
    payment_method_types: ['card'],
    line_items: [{
      price_data: {
        currency: opportunite.devise.toLowerCase(),
        product_data: { name: opportunite.titre },
        unit_amount: Math.round(opportunite.montantEstime * 100),
      },
      quantity: 1,
    }],
    mode: 'payment',
```

```
    success_url: `${process.env.NEXTAUTH_URL}/opportunites?success=true`,
    cancel_url: `${process.env.NEXTAUTH_URL}/opportunites?canceled=true`,
    metadata: { opportuniteId: opportunite.id },
  });

  return session;
}
```

Webhook Stripe

```
// app/api/webhooks/stripe/route.ts
export async function POST(request: NextRequest) {
  const body = await request.text();
  const signature = request.headers.get('stripe-signature')!;

  const event = stripe.webhooks.constructEvent(
    body,
    signature,
    process.env.STRIPE_WEBHOOK_SECRET!
  );

  if (event.type === 'checkout.session.completed') {
    const session = event.data.object;
    const opportuniteId = session.metadata.opportuniteId;

    await prisma.opportunite.update({
      where: { id: opportuniteId },
      data: { statutPaiement: 'paye' },
    });
  }

  return NextResponse.json({ received: true });
}
```

Déploiement

Vercel

1. Connecter le repository GitHub à Vercel
2. Configurer les variables d'environnement
3. Déployer

Variables de production

```
DATABASE_URL="postgresql://..."
NEXTAUTH_SECRET="production-secret"
NEXTAUTH_URL="https://votre-domaine.com"
```

```
STRIPE_SECRET_KEY="sk_live_..."
STRIPE_WEBHOOK_SECRET="whsec_..."
```

Base de données

Recommandations :

- **Supabase** : PostgreSQL managé gratuit
- **Neon** : PostgreSQL serverless
- **Railway** : PostgreSQL simple

Bonnes pratiques

Code style

- TypeScript strict mode activé
- Interfaces plutôt que types
- Composants fonctionnels uniquement
- Noms de variables en français (camelCase)

Sécurité

- Validation Zod sur toutes les entrées
- Vérification de session sur chaque route API
- Filtrage par `proprietaireId` pour l'isolation des données
- Hash bcrypt pour les mots de passe

Performance

- React Query pour le cache côté client
- Prisma avec sélection des champs nécessaires
- Images optimisées avec Next.js Image
- Lazy loading des composants lourds

Tests

```
# Lancer les tests
npm run test

# Tests avec couverture
npm run test:coverage
```

Scripts npm

Commande	Description

Commande	Description
`npm run dev`	Serveur de développement
`npm run build`	Build de production
`npm run start`	Serveur de production
`npm run lint`	Vérification ESLint
`npm run db:generate`	Génère le client Prisma
`npm run db:push`	Applique le schéma à la BDD
`npm run db:studio`	Interface Prisma Studio
`npm run db:migrate`	Crée une migration

Document généré le 9 décembre 2025