Documentation Technique Complète - CORTEXIA

Version : 1.0.0 (MVP)
Date : 20 janvier 2026
Équipe : Développement CORTEXIA

Table des matières

Vue d'ensemble

Objectif du projet

CORTEXIA est un assistant intelligent de transcription et de compte-rendu de réunions en temps réel. L'application permet de :

- Capturer l'audio d'une réunion (micro ou système)
- Transcrire en temps réel (FR/EN)
- Générer automatiquement des résumés
- Extraire actions et décisions
- Produire des comptes-rendus et emails de suivi

État actuel : MVP Fonctionnel ✅

Le MVP (Minimum Viable Product) est **complètement fonctionnel** et déployable localement. Toutes les fonctionnalités de base sont implémentées avec des simulations intelligentes pour les parties nécessitant des APIs externes.

Architecture technique

Stack technique

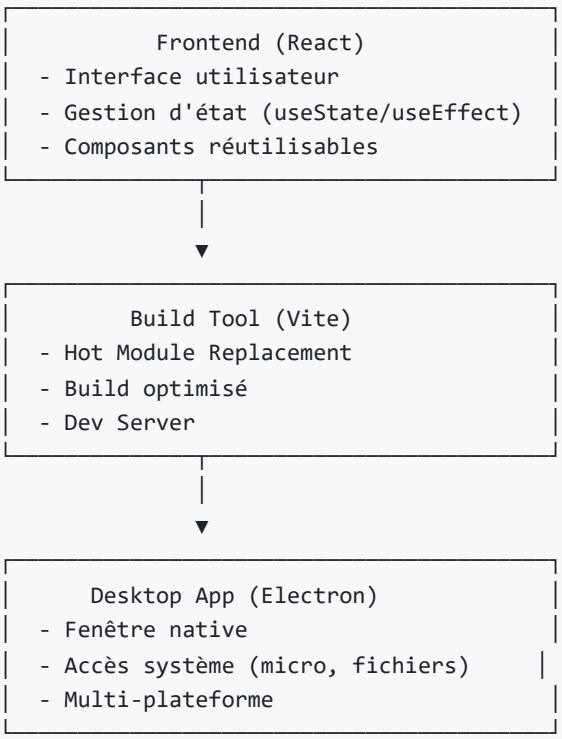
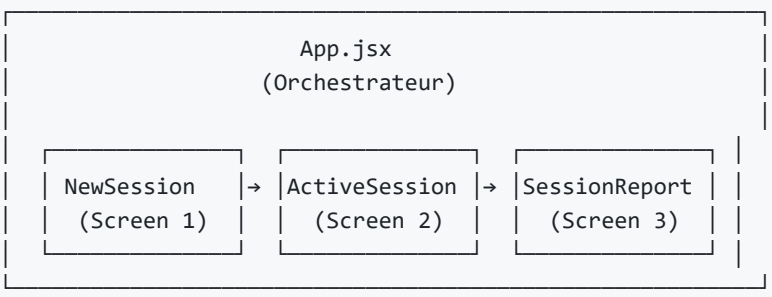


Schéma de l'application



Architecture en couches

```
┌─────────────────────────────────────────┐
│         Couche Présentation             │
│  (React Components + Styles)            │
├─────────────────────────────────────────┤
│         Couche Logique Métier           │
│  (Event Handlers, State Management)     │
├─────────────────────────────────────────┤
│         Couche Services                 │
│  (Audio, Transcription, LLM, Storage)   │
├─────────────────────────────────────────┤
│         Couche APIs/Natives             │
│  (Web Speech API, Navigator.mediaDevices)│
└─────────────────────────────────────────┘
```

Fonctionnalités implémentées

1. Écran "Nouvelle Session" (NewSession.jsx)

Fonctionnalités :

- ✅ Saisie du titre de réunion (validation requise)
- ✅ Sélection de la source audio (microphone/système)
- ✅ Choix de la langue (FR/EN)
- ✅ Case de consentement obligatoire (RGPD)
- ✅ Affichage des informations légales
- ✅ Validation du formulaire avant démarrage
- ✅ Interface responsive et accessible

Code principal :

```
// Validation et démarrage
const handleStart = () => {
  if (!consent) {
    alert('⚠️ Vous devez confirmer avoir le consentement');
    return;
  }
  if (!meetingTitle.trim()) {
    alert('⚠️ Veuillez donner un titre à la réunion');
    return;
  }
  onStart({ audioSource, language, title: meetingTitle });
};
```

États gérés :

- `audioSource` : 'microphone' | 'system'
- `language` : 'fr' | 'en'
- `consent` : boolean
- `meetingTitle` : string

---

2. Écran "Session Active" (ActiveSession.jsx)

Fonctionnalités :

- ✅ Capture audio en temps réel (Web Media API)
- ✅ Transcription en temps réel (Web Speech API)
- ✅ Affichage du temps écoulé (format HH:MM:SS)
- ✅ Défilement automatique de la transcription
- ✅ Pause/Reprise de l'enregistrement
- ✅ Marquage de moments importants avec note personnalisée
- ✅ Statistiques en direct (segments, moments marqués)
- ✅ Indicateur visuel d'enregistrement (point rouge animé)
- ✅ Confirmation avant arrêt de session
- ✅ Gestion des erreurs et permissions micro

Capture audio :

```
const stream = await navigator.mediaDevices.getUserMedia({
  audio: {
    echoCancellation: true,    // Réduction d'écho
    noiseSuppression: true,    // Suppression du bruit
    autoGainControl: true      // Contrôle automatique du gain
  }
});
```

Transcription en temps réel :

```
const SpeechRecognition = window.SpeechRecognition ||
                          window.webkitSpeechRecognition;
const recognition = new SpeechRecognition();
recognition.continuous = true;        // Transcription continue
recognition.interimResults = true;    // Résultats intermédiaires
recognition.lang = language === 'fr' ? 'fr-FR' : 'en-US';
```

```
  recognition.onresult = (event) => {
    const last = event.results.length - 1;
    const text = event.results[last][0].transcript;
    const isFinal = event.results[last].isFinal;

    if (isFinal) {
      setTranscript(prev => [...prev, {
        id: Date.now(),
        timestamp: Date.now(),
        text: text.trim(),
        speaker: 'Participant',
        isFinal: true
      }]);
    }
  };
```

États gérés :

- `transcript` : Array
- `duration` : number (secondes)
- `isRecording` : boolean
- `isPaused` : boolean
- `mediaRecorderRef` : MediaRecorder
- `recognitionRef` : SpeechRecognition

Structure d'une ligne de transcription :

```
interface TranscriptLine {
  id: number;
  timestamp: number;
  text: string;
  speaker: string;
  isFinal?: boolean;
  marked?: boolean;
  isSystem?: boolean;
}
```

---

3. Écran "Compte-rendu" (SessionReport.jsx)

Fonctionnalités :

- ✅ Génération automatique du résumé (actuellement simulé)
- ✅ Extraction d'actions avec responsables et échéances
- ✅ Identification des décisions prises

- ✅ Génération d'email de suivi pré-rédigé
- ✅ Interface à onglets (Résumé/Actions/Décisions/Email)
- ✅ Export en Markdown (.md)
- ✅ Export de la transcription brute (.txt)
- ✅ Copie de l'email dans le presse-papier
- ✅ Métadonnées de session (durée, langue, segments)
- ✅ Animation de chargement pendant génération

Génération du rapport (actuellement simulé) :

```
const generateReport = async () => {
  setTimeout(() => {
    // Simulation - À remplacer par appel API LLM
    setSummary(`📋 Résumé de la réunion...`);
    setActions([...]);
    setDecisions([...]);
    setFollowUpEmail(`Email pré-rédigé...`);
    setIsGenerating(false);
  }, 2500);
};
```

Structure des données :

```
interface Action {
  id: number;
  task: string;
  responsible: string;
  deadline: string;  // Format YYYY-MM-DD
  priority: 'Haute' | 'Moyenne' | 'Basse';
}

interface Decision {
  id: number;
  text: string;
  impact: 'Technique' | 'Sécurité' | 'Fonctionnel' | 'Légal';
}

interface SessionData {
  transcript: TranscriptLine[];
  duration: number;
  sessionId: string;
  language: 'fr' | 'en';
  endTime: number;
}
```

Export Markdown :

```
const content = `# Compte-rendu de réunion
${new Date().toLocaleDateString('fr-FR')}

${summary}

## Décisions
${decisions.map(d => `- **${d.text}** (Impact: ${d.impact})`).join('\n')}

## Actions à suivre
${actions.map(a => `- [ ] **${a.task}**
  - Responsable: ${a.responsible}
  - Échéance: ${a.deadline}
  - Priorité: ${a.priority}`).join('\n\n')}`;

const blob = new Blob([content], { type: 'text/markdown' });
// Téléchargement automatique
```

Structure du code

```
cortexia/
├── package.json                 # Dépendances et scripts
├── vite.config.js               # Configuration Vite
├── index.html                   # Point d'entrée HTML
├── README.md                    # Documentation utilisateur
├── .gitignore                   # Exclusions Git
│
├── electron/
│   └── main.js                  # Process principal Electron
│                                # - Création de fenêtre
│                                # - Gestion IPC
│                                # - Détection dev/prod
│
└── src/
    ├── main.jsx                 # Point d'entrée React
    ├── App.jsx                  # Composant racine + routing
    │
    ├── components/
    │   ├── NewSession.jsx       # Écran 1 : Configuration
    │   ├── ActiveSession.jsx    # Écran 2 : Transcription live
    │   └── SessionReport.jsx    # Écran 3 : Compte-rendu
    │
    └── styles/
        └── app.css              # Styles globaux (CSS Variables)
```

Détail des fichiers clés

electron/main.js

Responsabilités :

- Création de la fenêtre Electron
- Chargement de l'URL dev ou du build
- Gestion du cycle de vie de l'app
- Handlers IPC pour les fonctionnalités futures

Code important :

```
const isDev = !app.isPackaged;

if (isDev) {
  mainWindow.loadURL('http://localhost:5173');
  mainWindow.webContents.openDevTools();
} else {
  mainWindow.loadFile(path.join(__dirname, '../dist/index.html'));
}
```

src/App.jsx

Responsabilités :

- Gestion de l'état global de l'application
- Navigation entre les 3 écrans
- Transmission des données entre composants
- Structure du layout (header, main, footer)

États principaux :

```
const [screen, setScreen] = useState('new'); // 'new' | 'active' | 'report'
const [sessionData, setSessionData] = useState(null);
const [reportData, setReportData] = useState(null);
```

src/styles/app.css

Organisation :

- Variables CSS (couleurs, ombres, espacements)
- Styles de base (reset, typographie)
- Styles par composant (layout, forms, buttons)
- Animations (fadeIn, pulse, slideIn, spin)
- Responsive design (media queries)

Variables CSS :

```
:root {
  --primary: #3498db;
  --danger: #e74c3c;
  --success: #27ae60;
  --shadow: 0 2px 8px rgba(0,0,0,0.1);
  /* ... */
}
```

Technologies utilisées

Frontend

- **React 18.2.0** - Library UI avec hooks
- **Vite 5.4.21** - Build tool ultra-rapide
- **CSS3** - Styles natifs avec variables CSS

Desktop

- **Electron 28.1.0** - Framework multi-plateforme
- **concurrently 8.2.2** - Lancement dev server + Electron
- **wait-on 7.2.0** - Attente du serveur Vite

APIs natives

- **Web Speech API** - Transcription en temps réel (chrome/edge)
- **MediaDevices API** - Capture audio microphone
- **Clipboard API** - Copie dans le presse-papier
- **Blob API** - Génération de fichiers à télécharger

Outils de développement

- **Node.js 24.13.0** - Runtime JavaScript
- **npm 11.6.2** - Gestionnaire de packages

APIs et dépendances

Dépendances de production

```
{
  "react": "^18.2.0",           // Library UI
  "react-dom": "^18.2.0",       // Rendu DOM
  "axios": "^1.6.2"             // Client HTTP (pour futures APIs)
}
```

Dépendances de développement

```
{
  "@vitejs/plugin-react": "^4.2.1",  // Plugin Vite pour React
  "electron": "^28.1.0",              // Framework desktop
  "vite": "^5.0.8",                   // Build tool
  "concurrently": "^8.2.2",           // Multi-commandes
  "wait-on": "^7.2.0"                 // Attente de ressource
}
```

APIs externes (à intégrer)

Aucune API externe n'est actuellement utilisée, ce qui rend l'app entièrement fonctionnelle offline (hors transcription qui nécessite une connexion pour Web Speech API).

APIs recommandées pour production :

- **Whisper API** (OpenAI) - Transcription professionnelle
- **Deepgram** - Alternative transcription temps réel
- **GPT-4 API** (OpenAI) - Génération résumés/actions
- **Claude API** (Anthropic) - Alternative LLM

Flux de données

Flux principal de l'application

```
1. DÉMARRAGE
   ├─> Utilisateur remplit le formulaire (NewSession)
   ├─> Validation (titre + consentement)
   └─> App.handleStartSession()
       ├─> Création sessionData
       └─> Navigation vers 'active'
```

```
2. SESSION ACTIVE
   ├─> Demande permission microphone
   ├─> Initialisation MediaRecorder
   ├─> Initialisation SpeechRecognition
   ├─> Boucle transcription
   │   ├─> recognition.onresult
   │   └─> setTranscript([...prev, newLine])
   ├─> Actions utilisateur (pause, marquer)
   └─> Clic "Terminer"
       ├─> stopRecording()
       └─> App.handleEndSession(transcript, duration)

3. GÉNÉRATION RAPPORT
   ├─> Navigation vers 'report'
   ├─> useEffect() → generateReport()
   ├─> Simulation traitement (2.5s)
   │   ├─> Analyse transcription (TODO: API)
   │   ├─> Génération résumé (TODO: API)
   │   ├─> Extraction actions (TODO: API)
   │   └─> Génération email
   └─> Affichage des résultats

4. EXPORTS & FIN
   ├─> Export MD/TXT (Blob + download)
   ├─> Copie email (navigator.clipboard)
   └─> Nouvelle session → Reset state
```

Flux de données entre composants

```
App.jsx (State central)
  │
  ├─> sessionData: { audioSource, language, title, startTime, id }
  │   └─> Passé à ActiveSession via props
  │
  ├─> reportData: { transcript, duration, sessionId, language, endTime }
  │   └─> Passé à SessionReport via props
  │
  └─> Callbacks:
      ├─> handleStartSession(config)
      ├─> handleEndSession(transcript, duration)
      └─> handleNewSession()
```

Guide d'installation

Prérequis

- **Node.js 18+** (testé avec v24.13.0)
- **npm 8+** (testé avec v11.6.2)

- **Windows 10/11** (testé) ou macOS/Linux

Installation complète

```
# 1. Cloner ou extraire le projet
cd C:\Users\Utilisateur\Downloads\cortexia

# 2. Installer les dépendances
npm install

# 3. Lancer en développement
npm start
```

Scripts disponibles

```
{
  "dev": "vite",                    // Serveur Vite seul
  "build": "vite build",            // Build de production
  "electron": "electron .",         // Electron seul
  "start": "concurrently..."        // Dev complet (Vite + Electron)
}
```

Configuration PowerShell (Windows)

Si erreur "execution de scripts est désactivée" :

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Développement

Lancer en mode développement

```
npm start
```

Cela lance :

1. **Vite** sur http://localhost:5173 (avec HMR)

2. **Electron** qui charge automatiquement Vite

Structure de développement

```
Terminal 1: Vite Dev Server
  ├─> Hot Module Replacement
  ├─> Fast Refresh React
  └─> Port 5173

Terminal 2: Electron
  ├─> Charge http://localhost:5173
  ├─> DevTools ouvertes
  └─> Rechargement auto si Vite change
```

Modifier le code

Interface :

- Modifier `src/components/*.jsx` → Vite recharge auto
- Modifier `src/styles/app.css` → Styles mis à jour instantanément

Electron :

- Modifier `electron/main.js` → Relancer `npm start`

Debugging

Console navigateur (DevTools) :

- Ouverte automatiquement dans Electron
- Affiche les erreurs React/JavaScript
- Console.log visible

Erreurs communes :

1. **Port 5173 occupé** → Vite prend automatiquement 5174
2. **Microphone refusé** → Vérifier permissions système
3. **Web Speech API non supporté** → Utiliser Chrome/Edge
4. **Module non trouvé** → `npm install` puis relancer

Ce qui reste à faire

Sprint 2 : Production-Ready (2 semaines)

1. Intégration API de transcription professionnelle

Priorité : HAUTE
Temps estimé : 3-4 jours

Tâches :

- [ ] Choisir l'API (Whisper/Deepgram/AssemblyAI)
- [ ] Créer compte et obtenir clé API
- [ ] Créer `src/services/transcription.js`
- [ ] Remplacer Web Speech API dans ActiveSession.jsx
- [ ] Gérer le streaming audio vers l'API
- [ ] Implémenter gestion des erreurs réseau
- [ ] Ajouter retry logic
- [ ] Tester qualité FR/EN

Code à créer :

```
// src/services/transcription.js
export class TranscriptionService {
  constructor(apiKey, language) {
    this.apiKey = apiKey;
    this.language = language;
  }

  async startTranscription(audioStream) {
    // Connexion WebSocket Deepgram
    // ou POST chunks vers Whisper API
  }

  onTranscript(callback) {
    // Émission des transcriptions
  }

  stop() {
    // Nettoyage
  }
}
```

2. Intégration LLM pour résumés intelligents

Priorité : HAUTE
Temps estimé : 3-4 jours

Tâches :

- [ ] Choisir l'API (GPT-4/Claude/Gemini)

- [ ] Créer `src/services/llmService.js`
- [ ] Créer prompts optimisés pour :
  - Résumé de réunion
  - Extraction d'actions
  - Identification de décisions
  - Génération d'email
- [ ] Remplacer simulation dans SessionReport.jsx
- [ ] Implémenter streaming des réponses
- [ ] Ajouter indicateur de progression
- [ ] Gérer les erreurs API

Prompts recommandés :

```
const PROMPTS = {
  summary: `Tu es un assistant spécialisé dans les comptes-rendus de réunions.

Transcription:
{transcript}

Génère un résumé structuré avec:
- Points clés discutés (3-5 points)
- Contexte général
- Prochaines étapes

Format: Markdown`,

  actions: `Extrait toutes les actions/tâches de cette transcription.

Transcription:
{transcript}

Format JSON:
[{
  "task": "description",
  "responsible": "personne mentionnée ou 'Non assigné'",
  "deadline": "date si mentionnée ou 'À définir'",
  "priority": "Haute|Moyenne|Basse"
}]`
};
```

3. Stockage SQLite des sessions

Priorité : MOYENNE
Temps estimé : 2-3 jours

Tâches :

- [ ] Ajouter `better-sqlite3` (nécessite build tools)
- [ ] Créer `src/services/storage.js`
- [ ] Créer schéma de base de données
- [ ] Implémenter CRUD sessions
- [ ] Ajouter écran "Historique"
- [ ] Recherche de sessions
- [ ] Chiffrement optionnel (crypto-js)

Schéma DB :

```
CREATE TABLE sessions (
  id TEXT PRIMARY KEY,
  title TEXT NOT NULL,
  date INTEGER NOT NULL,
  duration INTEGER,
  language TEXT,
  transcript TEXT,  -- JSON
  summary TEXT,
  actions TEXT,     -- JSON
  decisions TEXT,   -- JSON
  created_at INTEGER DEFAULT (strftime('%s', 'now'))
);

CREATE INDEX idx_sessions_date ON sessions(date DESC);
CREATE INDEX idx_sessions_title ON sessions(title);
```

4. Export PDF professionnel

Priorité : MOYENNE
Temps estimé : 2 jours

Tâches :

- [ ] Intégrer jsPDF + jsPDF-autotable
- [ ] Créer template PDF branded
- [ ] Ajouter logo/couleurs entreprise
- [ ] Mise en page professionnelle
- [ ] Table des matières cliquable
- [ ] Génération PDF depuis SessionReport

Bibliothèques :

```
npm install jspdf jspdf-autotable
```

5. Support multi-locuteurs

Priorité : BASSE
Temps estimé : 3-4 jours

Tâches :

- [ ] Intégrer diarization API (Deepgram/AssemblyAI)
- [ ] Afficher speaker différemment dans transcript
- [ ] Permettre renommage des speakers
- [ ] Statistiques par speaker

6. Paramètres avancés

Priorité : BASSE
Temps estimé : 2 jours

Tâches :

- [ ] Écran de paramètres
- [ ] Niveau de verbosité résumé
- [ ] Templates personnalisables
- [ ] Thème sombre/clair
- [ ] Raccourcis clavier

🐛 Problèmes connus

1. Web Speech API limitations

Symptôme : Transcription parfois imprécise ou s'arrête
Cause : API gratuite limitée du navigateur
Solution temporaire : Parler clairement, proche du micro
Solution définitive : Intégrer Whisper/Deepgram (Sprint 2)

2. Pas de support Firefox

Symptôme : "Web Speech API non supportée"
Cause : Firefox n'implémente pas SpeechRecognition

Solution : Utiliser Chrome/Edge ou intégrer API externe

3. Résumé simulé

Symptôme : Le résumé est générique
Cause : Pas encore d'intégration LLM
Solution : Sprint 2 - Intégration GPT-4/Claude

4. Pas de sauvegarde des sessions

Symptôme : Sessions perdues au redémarrage
Cause : Pas de base de données
Solution : Sprint 2 - SQLite

5. Cache Electron warnings

Symptôme : Erreurs "Unable to move cache" en console
Cause : Permissions Windows
Impact : Aucun, warnings ignorables
Solution : Ajouter `--disable-gpu-sandbox` si problématique

🚀 Améliorations futures

Phase 1 : Production (Sprint 2-3)

- ✅ APIs professionnelles
- ✅ Stockage persistant
- ✅ Export PDF
- ✅ Packaging (.exe, .dmg, .deb)

Phase 2 : Collaboration (Sprint 4-5)

- ☐ Synchronisation cloud (optionnelle)
- ☐ Partage de sessions
- ☐ Annotations collaboratives
- ☐ Intégrations (Slack, Teams, Notion)

Phase 3 : Intelligence (Sprint 6-7)

- ☐ Analyse sentiment
- ☐ Détection de topics automatique

- [ ] Suggestions proactives
- [ ] Templates intelligents par type de réunion

Phase 4 : Enterprise (Sprint 8+)

- [ ] SSO (SAML, OAuth)
- [ ] Administration multi-utilisateurs
- [ ] Conformité renforcée (SOC2, ISO27001)
- [ ] Audit logs détaillés
- [ ] API pour intégrations tierces

📊 Métriques et KPIs

Métriques techniques

- **Temps de chargement initial :** < 2s
- **Latence transcription :** < 500ms (Web Speech)
- **Taux d'erreur transcription :** ~10-15% (dépend de l'audio)
- **Build size :** ~5MB (non optimisé)
- **RAM usage :** ~150MB (Electron + Renderer)

Métriques à suivre en production

- Taux de complétion des sessions
- Durée moyenne des sessions
- Nombre d'exports par session
- Utilisation des fonctionnalités (pause, marquer)
- Erreurs de transcription signalées

🔐 Sécurité et conformité

Implémenté

- ✅ Consentement explicite avant enregistrement
- ✅ Stockage local uniquement
- ✅ Pas de télémétrie
- ✅ Bandeau d'avertissement pendant transcription

À implémenter (Sprint 2)

- [ ] Chiffrement des données au repos
- [ ] Chiffrement des communications API
- [ ] Journal d'audit
- [ ] Gestion des clés API sécurisée (.env)
- [ ] Politique de rétention configurable
- [ ] Export/suppression des données (RGPD)

📞 Support et contribution

Rapporter un bug

1. Vérifier les Problèmes connus
2. Ouvrir une issue GitHub avec :
   - Description du problème
   - Étapes de reproduction
   - Logs de la console
   - Système d'exploitation

Contribuer

1. Fork le projet
2. Créer une branche (`git checkout -b feature/AmazingFeature`)
3. Commit (`git commit -m 'Add AmazingFeature'`)
4. Push (`git push origin feature/AmazingFeature`)
5. Ouvrir une Pull Request

📄 Licence

Dernière mise à jour : 20 janvier 2026
Mainteneur : Équipe CORTEXIA
Version documentation : 1.0.0

🎨 Identité Visuelle

Logo

- **Fichier :** `src/assets/logo.svg`
- **Design :** Cerveau avec circuits technologiques
- **Couleurs :** Gradient bleu (#1e88e5) → violet (#8e24aa)
- **Usage :** Header application, documentation, marketing

Charte graphique

- **Couleur principale :** #3498db (bleu professionnel)
- **Couleur secondaire :** #8e24aa (violet innovation)
- **Police :** System fonts (Segoe UI, Roboto)
- **Style :** Moderne, technologique, professionnel