📋 Consultator

Documentation Complète

Application de gestion d'une practice data de consultants

Interface Streamlit moderne avec analyses avancées

Version 1.0.0

Générée automatiquement depuis Sphinx

Septembre 2025

Table des matières

Page de garde

📋 Consultator - Documentation Complète

:alt: Logo Consultator

:align: center

:width: 200px

|

\*\*Consultator\*\* est une application Streamlit moderne pour la gestion d'une practice data de 60 consultants.

Cette documentation fournit tous les guides nécessaires pour comprendre, installer, utiliser et contribuer au projet.

📖 \*\*Premiers pas\*\*

:gutter: 3

.. grid-item-card:: 🚀 Installation

:link: installation

:link-type: doc

Guide complet d'installation et configuration

.. grid-item-card:: 🎯 Démarrage rapide

:link: quickstart

:link-type: doc

Lancez-vous en 5 minutes

.. grid-item-card:: 📚 Tutoriels

:link: tutorials/index

:link-type: doc

Guides pas à pas détaillés

.. grid-item-card:: 🔧 API Reference

:link: api/index

:link-type: doc

Documentation technique complète

🔍 \*\*Fonctionnalités principales\*\*

:gutter: 2

.. grid-item-card:: 👥 Gestion Consultants

:link: features/consultants

:link-type: doc

CRUD complet, profils, compétences

.. grid-item-card:: 🤖 Chatbot IA

:link: features/chatbot

:link-type: doc

Assistant intelligent pour les données

.. grid-item-card:: 📊 Analytics

:link: features/analytics

:link-type: doc

Tableaux de bord et métriques

.. grid-item-card:: 📄 Documents

:link: features/documents

:link-type: doc

Parsing CV, génération rapports

.. grid-item-card:: 👔 Business Managers

:link: features/business\_managers

:link-type: doc

Gestion des managers métier

.. grid-item-card:: 🧪 Tests & Qualité

:link: development/testing

:link-type: doc

Tests automatisés et CI/CD

📋 \*\*Table des matières\*\*

:maxdepth: 2

:caption: 🚀 Premiers pas

installation

quickstart

:maxdepth: 2

:caption: 🎯 Fonctionnalités

features

:maxdepth: 2

:caption: 🔧 Développement

development

:maxdepth: 2

:caption: 📚 API & Référence

api

:maxdepth: 2

:caption: 📖 Guides & Tutoriels

tutorials

📊 \*\*Statistiques du projet\*\*

.. list-table:: Métriques clés

:header-rows: 1

:widths: 30 20 20 30

\* - Métrique

- Valeur

- Statut

- Description

\* - Lignes de code

- ~15,000

- ✅ Stable

- Code Python + configuration

\* - Fichiers Python

- 49

- ✅ Organisé

- Architecture modulaire

\* - Tests unitaires

- 407

- ✅ 97.8% réussite

- Couverture 26%

\* - Dépendances

- 25+

- ✅ Gérées

- requirements.txt complet

🔗 \*\*Liens utiles\*\*

.. list-table:: Ressources externes

:header-rows: 1

\* - Ressource

- URL

- Description

\* - Repository GitHub

- https://github.com/ericfunman/Consultator

- Code source et issues

\* - SonarCloud

- https://sonarcloud.io/dashboard?id=ericfunman\_Consultator

- Qualité et sécurité du code

\* - Documentation

- https://ericfunman.github.io/Consultator/

- Version déployée (GitHub Pages)

📞 \*\*Support & Contact\*\*

- \*\*Discussions\*\*: `https://github.com/ericfunman/Consultator/discussions`

|

💡 Cette documentation est générée automatiquement à partir du code source et des docstrings. Elle est mise à jour automatiquement lors de chaque push sur la branche principale.

💡 Utilisez la fonction de recherche (🔍) en haut à droite pour trouver rapidement ce dont vous avez besoin.

\*\*Dernière mise à jour:\*\* |today| | \*\*Version:\*\* 1.0.0

Installation et Configuration

Cette section explique comment installer et configurer Consultator sur votre système.

Prérequis système

\*\*Python\*\*

- Version 3.8 ou supérieure

- Recommandé : Python 3.11 ou 3.12

\*\*Système d'exploitation\*\*

- Windows 10/11

- macOS 12+

- Linux (Ubuntu 20.04+, CentOS 8+)

\*\*Mémoire RAM\*\*

- Minimum : 4 GB

- Recommandé : 8 GB ou plus

Installation rapide

\*\*Cloner le repository\*\*

.. code-block:: bash

git clone https://github.com/votre-organisation/consultator.git

cd consultator

\*\*Créer un environnement virtuel\*\*

.. code-block:: bash

python -m venv venv

# Windows

venv\Scripts\activate

# Linux/macOS

source venv/bin/activate

\*\*Installer les dépendances\*\*

.. code-block:: bash

pip install -r requirements.txt

4. \*\*Initialiser la base de données\*\*

.. code-block:: bash

python -c "from app.database.database import init\_database; init\_database()"

5. \*\*Lancer l'application\*\*

.. code-block:: bash

streamlit run run.py

Installation détaillée

Configuration de l'environnement de développement

Pour les développeurs souhaitant contribuer au projet :

# Installation des dépendances de développement  
pip install -r requirements-dev.txt

# Installation des outils de qualité

pip install -r requirements-test.txt

# Configuration de pre-commit

pre-commit install

Configuration de la base de données

La base de données SQLite est créée automatiquement. Pour une configuration personnalisée :

# Dans config/settings.py  
DATABASE\_URL = "sqlite:///data/consultator.db"

Configuration des variables d'environnement

Créer un fichier ``.env`` à la racine du projet :

# Configuration de l'application  
APP\_ENV=development  
DEBUG=True

# Configuration de la base de données

DATABASE\_URL=sqlite:///data/consultator.db

# Configuration du logging

LOG\_LEVEL=INFO

LOG\_FILE=logs/consultator.log

Dépannage

Problèmes courants et solutions :

\*\*Erreur d'importation de modules\*\*

Vérifier que l'environnement virtuel est activé et que toutes les dépendances sont installées.

\*\*Erreur de base de données\*\*

Supprimer le fichier de base de données et relancer l'initialisation :

.. code-block:: bash

rm data/consultator.db

python -c "from app.database.database import init\_database; init\_database()"

\*\*Problèmes de performance\*\*

- Vérifier la version de Python (3.11+ recommandé)

- Augmenter la mémoire RAM si nécessaire

- Utiliser un SSD pour le stockage

Mise à jour

Pour mettre à jour Consultator :

# Récupérer les dernières modifications  
git pull origin main

# Mettre à jour les dépendances

pip install -r requirements.txt --upgrade

# Migrer la base de données si nécessaire

python scripts/migrate\_database.py

Support

En cas de problème d'installation :

- Ouvrir une `issue sur GitHub <https://github.com/votre-organisation/consultator/issues>`\_

Guide de démarrage rapide

Bienvenue dans Consultator ! Ce guide vous permettra de prendre en main l'application en quelques minutes.

Premiers pas

\*\*Démarrer l'application\*\*

Après l'installation, lancez Consultator :

.. code-block:: bash

streamlit run run.py

L'application s'ouvrira dans votre navigateur à l'adresse ``http://localhost:8501``.

\*\*Interface principale\*\*

L'application se compose de plusieurs sections accessibles via le menu latéral :

- \*\*🏠 Accueil\*\* : Vue d'ensemble et métriques

- \*\*👥 Consultants\*\* : Gestion des profils

- \*\*📊 Analyses\*\* : Tableaux de bord et rapports

- \*\*⚙️ Configuration\*\* : Paramètres système

Créer votre premier consultant

\*\*Accéder à la gestion des consultants\*\*

Cliquez sur "👥 Consultants" dans le menu latéral.

\*\*Ajouter un consultant\*\*

Cliquez sur le bouton "➕ Ajouter un consultant".

\*\*Remplir le formulaire\*\*

.. code-block:: none

Informations personnelles :

- Nom : DUPONT

- Prénom : Jean

- Email : jean.dupont@consultant.com

- Téléphone : +33 1 23 45 67 89

Compétences techniques :

- Python : 5 ans d'expérience

- SQL : 3 ans d'expérience

- Machine Learning : 2 ans d'expérience

4. \*\*Sauvegarder\*\*

Cliquez sur "💾 Sauvegarder" pour enregistrer le profil.

Importer des données

\*\*Import depuis Excel\*\*

Préparer votre fichier Excel avec les colonnes suivantes :

.. list-table:: Format d'import Excel

:header-rows: 1

\* - nom

- prenom

- email

- telephone

- competence\_1

- annees\_exp\_1

\* - DUPONT

- Jean

- jean.dupont@email.com

- +33123456789

- Python

- 5

Aller dans "👥 Consultants" → "📥 Importer"

Sélectionner votre fichier Excel

4. Cliquer sur "🚀 Importer"

Explorer les analyses

\*\*Tableaux de bord\*\*

Accéder à "📊 Analyses"

Consulter les métriques principales :

- Nombre total de consultants

- Répartition par compétence

- Revenus moyens par consultant

- Évolution des missions

\*\*Filtres et recherche\*\*

- Filtrer par compétence, niveau d'expérience, ou statut

Configuration initiale

\*\*Paramètres système\*\*

Aller dans "⚙️ Configuration"

Configurer :

- \*\*Référentiel de compétences\*\* : Ajouter/modifier les compétences disponibles

- \*\*Paramètres d'import\*\* : Configurer les mappings Excel

- \*\*Préférences utilisateur\*\* : Thème, langue, notifications

\*\*Données de test\*\*

Pour découvrir les fonctionnalités, vous pouvez charger des données de test :

python scripts/generate\_test\_data.py

Fonctionnalités avancées

\*\*Gestion des missions\*\*

- Suivre les revenus et la durée des projets

\*\*Rapports personnalisés\*\*

- Exporter en PDF ou Excel

\*\*Intégration chatbot IA\*\*

- Obtenir des analyses instantanées

Prochaines étapes

Maintenant que vous maîtrisez les bases :

\*\*Explorez la documentation complète\*\* : `📚 Documentation <https://consultator.readthedocs.io/>`\_

\*\*Personnalisez votre installation\*\* : Configurez les compétences et paramètres

\*\*Importez vos données réelles\*\* : Migrez vos données existantes

4. \*\*Découvrez les analyses avancées\*\* : Créez vos premiers rapports

Support et communauté

- \*\*Issues GitHub\*\* : `Signaler un bug <https://github.com/votre-organisation/consultator/issues>`\_

N'hésitez pas à contacter l'équipe pour toute question !

Fonctionnalités

Consultator offre une suite complète d'outils pour la gestion d'une practice data de consultants.

Vue d'ensemble

:gutter: 3

:margin: 0

:padding: 3

.. grid-item-card:: 👥 Gestion des consultants

:link: features-consultants

:link-type: ref

Gestion complète des profils, compétences et parcours professionnels des consultants.

.. grid-item-card:: 📊 Analyses et rapports

:link: features-analytics

:link-type: ref

Tableaux de bord interactifs et rapports personnalisables pour le pilotage de l'activité.

.. grid-item-card:: 🤖 Chatbot IA

:link: features-chatbot

:link-type: ref

Assistant intelligent pour l'analyse des données et les recommandations métier.

.. grid-item-card:: 📥 Import/Export

:link: features-import

:link-type: ref

Intégration transparente avec Excel et autres formats de données.

Gestion des consultants

\*\*Profils détaillés\*\*

Chaque consultant dispose d'un profil complet incluant :

- \*\*Parcours professionnel\*\* : CV, expériences, formations

\*\*Disponibilité\*\* : Statut actuel, contraintes de mobilité

\*\*Gestion des compétences\*\*

- Niveaux d'expérience (débutant, intermédiaire, expert)

Évolution des compétences dans le temps

\*\*Suivi des missions\*\*

- Revenus et durée des projets

Compétences mises en œuvre

Analyses et rapports

\*\*Tableaux de bord en temps réel\*\*

Métriques clés accessibles instantanément :

- \*\*Compétences\*\* : Répartition par domaine, niveaux moyens, gaps identifiés

\*\*Performance\*\* : Taux d'occupation, satisfaction client, turnover

\*\*Rapports personnalisés\*\*

- Filtres avancés (date, compétence, statut, etc.)

Planification automatique des rapports

\*\*Analyses prédictives\*\*

- Identification des gaps de recrutement

Alertes sur les risques de turnover

Chatbot IA intégré

\*\*Assistant intelligent\*\*

Posez des questions en langage naturel :

- \*"Quel est le CA généré par la compétence Data Science ce trimestre ?"\*

\*\*Fonctionnalités avancées\*\*

- \*\*Recommandations\*\* : Suggestions de profils pour les missions

\*\*Alertes intelligentes\*\* : Notifications sur les événements importants

\*\*Apprentissage continu\*\*

Le chatbot s'améliore avec l'usage grâce à :

- Feedback utilisateur

Intégration de nouvelles sources de données

Import et export de données

\*\*Formats supportés\*\*

- \*\*CSV\*\* : Pour les données structurées simples

\*\*PDF\*\* : Extraction depuis les CV

\*\*Import intelligent\*\*

- \*\*Validation des données\*\* : Contrôles de cohérence et format

\*\*Mise à jour incrémentale\*\* : Ajout ou modification des données existantes

\*\*Export flexible\*\*

- \*\*Filtres personnalisés\*\* : Sélection des données à exporter

\*\*Archivage\*\* : Historique des exports avec métadonnées

Sécurité et confidentialité

\*\*Protection des données\*\*

- \*\*Contrôle d'accès\*\* : Gestion fine des permissions par utilisateur

\*\*RGPD compliant\*\* : Conformité aux réglementations européennes

\*\*Sauvegarde et récupération\*\*

- \*\*Récupération point-in-time\*\* : Restauration à une date précise

\*\*Tests de récupération\*\* : Validation régulière des procédures

Intégrations

\*\*Outils externes\*\*

- \*\*GitHub/GitLab\*\* : Synchronisation des contributions techniques

\*\*Slack/Teams\*\* : Notifications et intégration chatbot

\*\*APIs et webhooks\*\*

- \*\*Webhooks\*\* : Notifications temps réel des événements

\*\*Swagger/OpenAPI\*\* : Documentation automatique des APIs

Évolutivité et performance

\*\*Architecture scalable\*\*

- \*\*Cache\*\* : Redis pour les performances optimales

\*\*Microservices\*\* : Architecture modulaire pour l'évolutivité

\*\*Optimisations\*\*

- \*\*Indexation\*\* : Recherche instantanée sur tous les champs

\*\*Monitoring\*\* : Métriques de performance en temps réel

Guide de développement

Ce guide présente les bonnes pratiques et conventions de développement pour contribuer à Consultator.

Architecture du projet

Structure des dossiers

::

consultator/

├── app/ # Code principal de l'application

│ ├── database/ # Modèles et connexion base de données

│ ├── pages/ # Pages Streamlit

│ ├── services/ # Services métier

│ ├── utils/ # Utilitaires et helpers

│ └── components/ # Composants réutilisables

├── config/ # Configuration de l'application

├── tests/ # Tests unitaires et d'intégration

├── docs/ # Documentation Sphinx

├── scripts/ # Scripts utilitaires

└── data/ # Données et fichiers temporaires

Architecture en couches

\*\*Couche présentation (Streamlit)\*\*

Pages et composants utilisateur dans ``app/pages/``

\*\*Couche service\*\*

Logique métier dans ``app/services/``

\*\*Couche données\*\*

Modèles et accès base de données dans ``app/database/``

\*\*Couche utilitaires\*\*

Fonctions helpers dans ``app/utils/``

Standards de code

Style Python

\*\*PEP 8\*\*

Respect strict des conventions PEP 8 avec Black pour le formatage automatique.

\*\*Type hints\*\*

Utilisation systématique des annotations de type :

.. code-block:: python

from typing import List, Optional

def get\_consultant\_by\_id(consultant\_id: int) -> Optional[Consultant]:

"""Récupère un consultant par son ID."""

pass

\*\*Docstrings\*\*

Documentation complète en français :

.. code-block:: python

def calculate\_revenue(consultant: Consultant,

start\_date: date,

end\_date: date) -> float:

"""

Calcule le revenu généré par un consultant sur une période.

Args:

consultant: Instance du consultant

start\_date: Date de début de la période

end\_date: Date de fin de la période

Returns:

Revenu total en euros

Raises:

ValueError: Si les dates sont invalides

"""

pass

Nommage

.. list-table:: Conventions de nommage

:header-rows: 1

\* - Élément

- Convention

- Exemples

\* - Classes

- PascalCase

- ``ConsultantService``, ``DatabaseManager``

\* - Fonctions

- snake\_case

- ``get\_consultant\_by\_id``, ``calculate\_revenue``

\* - Variables

- snake\_case

- ``consultant\_list``, ``total\_revenue``

\* - Constantes

- UPPER\_SNAKE\_CASE

- ``DATABASE\_PATH``, ``MAX\_RETRIES``

\* - Modules

- snake\_case

- ``consultant\_service.py``, ``database\_utils.py``

Gestion d'erreurs

\*\*Exceptions personnalisées\*\*

class ConsultatorError(Exception):  
 """Exception de base pour Consultator."""  
 pass

class ValidationError(ConsultatorError):

"""Erreur de validation des données."""

pass

class DatabaseError(ConsultatorError):

"""Erreur de base de données."""

pass

\*\*Gestion contextuelle\*\*

from contextlib import contextmanager

@contextmanager

def database\_session():

"""Context manager pour les sessions de base de données."""

session = SessionLocal()

try:

yield session

session.commit()

except Exception as e:

session.rollback()

raise DatabaseError(f"Erreur base de données: {e}")

finally:

session.close()

\*\*Logging\*\*

import logging

logger = logging.getLogger(\_\_name\_\_)

def process\_consultant\_data(data: dict):

logger.info("Début du traitement des données consultant")

try:

# Traitement des données

logger.debug(f"Données reçues: {data.keys()}")

result = validate\_and\_save(data)

logger.info("Traitement terminé avec succès")

return result

except ValidationError as e:

logger.warning(f"Erreur de validation: {e}")

raise

except Exception as e:

logger.error(f"Erreur inattendue: {e}", exc\_info=True)

raise

Base de données

Modèles SQLAlchemy

\*\*Définition des modèles\*\*

from sqlalchemy import Column, Integer, String, DateTime, ForeignKey  
from sqlalchemy.orm import relationship  
from sqlalchemy.ext.declarative import declarative\_base

Base = declarative\_base()

class Consultant(Base):

\_\_tablename\_\_ = "consultants"

id = Column(Integer, primary\_key=True, index=True)

nom = Column(String(100), nullable=False)

prenom = Column(String(100), nullable=False)

email = Column(String(200), unique=True, nullable=False)

created\_at = Column(DateTime, default=datetime.utcnow)

# Relations

competences = relationship("CompetenceConsultant",

back\_populates="consultant")

missions = relationship("Mission", back\_populates="consultant")

\*\*Migrations\*\*

Utilisation d'Alembic pour les migrations :

# Générer une nouvelle migration  
alembic revision --autogenerate -m "Ajout table missions"

# Appliquer les migrations

alembic upgrade head

Requêtes optimisées

\*\*Eager loading\*\*

from sqlalchemy.orm import joinedload

def get\_consultant\_with\_competences(consultant\_id: int):

return session.query(Consultant).options(

joinedload(Consultant.competences)

).filter(Consultant.id == consultant\_id).first()

\*\*Pagination\*\*

def get\_consultants\_paginated(page: int = 1, per\_page: int = 20):  
 offset = (page - 1) \* per\_page  
 return session.query(Consultant).offset(offset).limit(per\_page).all()

Services métier

Pattern des services

\*\*Structure d'un service\*\*

from typing import List, Optional  
from app.database.models import Consultant

class ConsultantService:

@staticmethod

def get\_all\_consultants(page: int = 1, per\_page: int = 20) -> List[Consultant]:

"""Récupère tous les consultants avec pagination."""

with database\_session() as session:

return session.query(Consultant).offset(

(page - 1) \* per\_page

).limit(per\_page).all()

@staticmethod

def get\_consultant\_by\_id(consultant\_id: int) -> Optional[Consultant]:

"""Récupère un consultant par son ID."""

with database\_session() as session:

return session.query(Consultant).filter(

Consultant.id == consultant\_id

).first()

@staticmethod

def create\_consultant(data: dict) -> Consultant:

"""Crée un nouveau consultant."""

# Validation des données

validated\_data = ConsultantValidator.validate(data)

with database\_session() as session:

consultant = Consultant(\*\*validated\_data)

session.add(consultant)

return consultant

\*\*Injection de dépendances\*\*

from dependency\_injector import containers, providers

class Container(containers.DeclarativeContainer):

config = providers.Configuration()

database = providers.Singleton(DatabaseManager, config.database)

consultant\_service = providers.Factory(

ConsultantService,

database=database

)

Tests

Structure des tests

::

tests/

├── unit/ # Tests unitaires

│ ├── test\_consultant\_service.py

│ └── test\_validators.py

├── integration/ # Tests d'intégration

│ ├── test\_database\_integration.py

│ └── test\_api\_integration.py

├── fixtures/ # Données de test

│ ├── consultants.json

│ └── sample\_data.sql

└── conftest.py # Configuration pytest

Tests unitaires

import pytest  
from unittest.mock import Mock, patch  
from app.services.consultant\_service import ConsultantService

class TestConsultantService:

def test\_get\_consultant\_by\_id\_success(self, mock\_session):

# Arrange

expected\_consultant = Mock()

mock\_session.query.return\_value.filter.return\_value.first.return\_value = expected\_consultant

# Act

result = ConsultantService.get\_consultant\_by\_id(1)

# Assert

assert result == expected\_consultant

mock\_session.query.assert\_called\_once()

def test\_get\_consultant\_by\_id\_not\_found(self, mock\_session):

# Arrange

mock\_session.query.return\_value.filter.return\_value.first.return\_value = None

# Act

result = ConsultantService.get\_consultant\_by\_id(999)

# Assert

assert result is None

Tests d'intégration

import pytest  
from app.database.database import init\_database, reset\_database

@pytest.fixture(scope="session", autouse=True)

def setup\_test\_database():

"""Initialise la base de test au début de la session."""

init\_database()

yield

reset\_database()

def test\_consultant\_creation\_integration(test\_client, sample\_consultant\_data):

# Act

response = test\_client.post("/api/consultants", json=sample\_consultant\_data)

# Assert

assert response.status\_code == 201

data = response.get\_json()

assert data["nom"] == sample\_consultant\_data["nom"]

assert "id" in data

Couverture de code

Objectif de couverture : \*\*90% minimum\*\*

# Exécuter les tests avec couverture  
pytest --cov=app --cov-report=html --cov-report=term

# Générer le rapport HTML

open htmlcov/index.html

CI/CD

Pipeline GitHub Actions

Le pipeline comprend :

\*\*Linting\*\* : flake8, black, isort

\*\*Tests\*\* : pytest avec couverture

\*\*Sécurité\*\* : audit des dépendances

4. \*\*Documentation\*\* : build Sphinx

5. \*\*Déploiement\*\* : vers staging/production

Configuration locale

\*\*Pre-commit hooks\*\*

# Installation  
pre-commit install

# Exécution manuelle

pre-commit run --all-files

\*\*Qualité du code\*\*

# Analyse complète  
python run\_quality\_pipeline.py

# Tests uniquement

pytest

# Linting

flake8 app/ tests/

Déploiement

Environnements

- \*\*Staging\*\* : Environnement de pré-production

Variables d'environnement

# .env.development  
APP\_ENV=development  
DEBUG=True  
DATABASE\_URL=sqlite:///data/dev.db  
LOG\_LEVEL=DEBUG

# .env.production

APP\_ENV=production

DEBUG=False

DATABASE\_URL=postgresql://user:pass@host:5432/consultator

LOG\_LEVEL=INFO

Containerisation

\*\*Dockerfile\*\*

FROM python:3.11-slim

WORKDIR /app

# Installation des dépendances système

RUN apt-get update && apt-get install -y \

gcc \

&& rm -rf /var/lib/apt/lists/\*

# Installation des dépendances Python

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Copie du code

COPY . .

# Exposition du port

EXPOSE 8501

# Commande de démarrage

CMD ["streamlit", "run", "run.py", "--server.port=8501", "--server.address=0.0.0.0"]

\*\*Docker Compose\*\*

version: '3.8'  
services:  
 consultator:  
 build: .  
 ports:  
 - "8501:8501"  
 volumes:  
 - ./data:/app/data  
 environment:  
 - DATABASE\_URL=sqlite:///data/consultator.db  
 depends\_on:  
 - postgres

postgres:

image: postgres:15

environment:

POSTGRES\_DB: consultator

POSTGRES\_USER: consultator

POSTGRES\_PASSWORD: password

volumes:

- postgres\_data:/var/lib/postgresql/data

Monitoring

Logs

\*\*Configuration centralisée\*\*

import logging.config

LOGGING\_CONFIG = {

'version': 1,

'formatters': {

'detailed': {

'format': '%(asctime)s - %(name)s - %(levelname)s - %(message)s'

}

},

'handlers': {

'file': {

'class': 'logging.FileHandler',

'filename': 'logs/consultator.log',

'formatter': 'detailed'

},

'console': {

'class': 'logging.StreamHandler',

'formatter': 'detailed'

}

},

'root': {

'level': 'INFO',

'handlers': ['file', 'console']

}

}

logging.config.dictConfig(LOGGING\_CONFIG)

Métriques

\*\*Performance\*\*

- Utilisation CPU/Mémoire

Taux d'erreur

\*\*Métier\*\*

- CA généré

Temps de traitement des imports

Alertes

\*\*Seuils critiques\*\*

- Temps de réponse > 5 secondes

Échec des sauvegardes

Contribution

Workflow Git

\*\*Créer une branche\*\*

.. code-block:: bash

git checkout -b feature/nouvelle-fonctionnalite

\*\*Développer\*\*

- Écrire des tests

- Implémenter la fonctionnalité

- Respecter les standards de code

\*\*Commiter\*\*

.. code-block:: bash

git add .

git commit -m "feat: ajout nouvelle fonctionnalité

- Description détaillée

- Tests ajoutés

- Documentation mise à jour"

4. \*\*Créer une PR\*\*

- Push de la branche

- Création de la Pull Request

- Revue par les pairs

Types de commits

- ``fix:`` Correction de bug

- ``style:`` Changement de style (formatage, etc.)

- ``test:`` Ajout ou modification de tests

Code review

\*\*Checklist\*\*

- [ ] Code respecte les standards PEP 8

- [ ] Documentation à jour

- [ ] Performance acceptable

Support

\*\*Ressources\*\*

- \*\*Slack\*\* : Canal #dev pour les questions techniques

\*\*Points de contact\*\*

- \*\*DevOps\*\* : marie.martin@consultator.com

API Reference

Cette section documente l'architecture API de Consultator, incluant les services métier et l'API REST (future).

Architecture des services

Consultator utilise une architecture en couches avec des services métier dédiés.

Services métier

:members:

:undoc-members:

:show-inheritance:

:members:

:undoc-members:

:show-inheritance:

:members:

:undoc-members:

:show-inheritance:

:members:

:undoc-members:

:show-inheritance:

:members:

:undoc-members:

:show-inheritance:

Modèles de données

:members:

:undoc-members:

:show-inheritance:

Utilitaires

:members:

:undoc-members:

:show-inheritance:

:members:

:undoc-members:

:show-inheritance:

API REST (Future)

Une API REST complète sera disponible dans les prochaines versions.

Endpoints prévus

\*\*Consultants\*\*

Liste tous les consultants.

\*\*Exemple de requête\*\*

.. code-block:: http

GET /api/v1/consultants?page=1&per\_page=20 HTTP/1.1

Host: api.consultator.com

Authorization: Bearer <token>

\*\*Paramètres de requête\*\*

- \*\*page\*\* (int) -- Numéro de page (défaut: 1)

- \*\*per\_page\*\* (int) -- Nombre d'éléments par page (défaut: 20)

- \*\*search\*\* (str) -- Terme de recherche

- \*\*skill\*\* (str) -- Filtre par compétence

\*\*Exemple de réponse\*\*

.. code-block:: json

{

"data": [

{

"id": 1,

"nom": "DUPONT",

"prenom": "Jean",

"email": "jean.dupont@consultant.com",

"competences": [

{

"nom": "Python",

"annees\_experience": 5,

"niveau": "Expert"

}

]

}

],

"pagination": {

"page": 1,

"per\_page": 20,

"total": 150,

"pages": 8

}

}

Crée un nouveau consultant.

\*\*Exemple de requête\*\*

.. code-block:: http

POST /api/v1/consultants HTTP/1.1

Host: api.consultator.com

Authorization: Bearer <token>

Content-Type: application/json

.. code-block:: json

{

"nom": "MARTIN",

"prenom": "Marie",

"email": "marie.martin@consultant.com",

"competences": [

{

"nom": "Data Science",

"annees\_experience": 3

}

]

}

Récupère les détails d'un consultant.

Met à jour un consultant.

Supprime un consultant.

\*\*Analyses\*\*

Récupère les métriques du tableau de bord.

\*\*Exemple de réponse\*\*

.. code-block:: json

{

"total\_consultants": 150,

"consultants\_actifs": 120,

"ca\_total": 2500000,

"competences\_top": [

{"nom": "Python", "count": 45},

{"nom": "SQL", "count": 38},

{"nom": "Machine Learning", "count": 32}

]

}

Génère un rapport spécifique.

\*\*Paramètres d'URL\*\*

- \*\*type\*\* (str) -- Type de rapport (revenue, skills, performance)

\*\*Authentification\*\*

Authentification utilisateur.

\*\*Exemple de requête\*\*

.. code-block:: json

{

"username": "admin",

"password": "password"

}

\*\*Exemple de réponse\*\*

.. code-block:: json

{

"access\_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",

"token\_type": "bearer",

"expires\_in": 3600

}

Rafraîchit le token d'accès.

Format des données

\*\*Consultant\*\*

{  
 "id": "integer",  
 "nom": "string",  
 "prenom": "string",  
 "email": "string",  
 "telephone": "string",  
 "date\_naissance": "date",  
 "adresse": "string",  
 "competences": [  
 {  
 "nom": "string",  
 "annees\_experience": "integer",  
 "niveau": "string",  
 "certifications": ["string"]  
 }  
 ],  
 "missions": [  
 {  
 "id": "integer",  
 "client": "string",  
 "poste": "string",  
 "date\_debut": "date",  
 "date\_fin": "date",  
 "tarif\_journalier": "number",  
 "statut": "string"  
 }  
 ],  
 "created\_at": "datetime",  
 "updated\_at": "datetime"  
}

\*\*Compétence\*\*

{  
 "id": "integer",  
 "nom": "string",  
 "categorie": "string",  
 "description": "string",  
 "niveau\_requis": "string"  
}

Codes d'erreur

.. list-table:: Codes d'erreur HTTP

:header-rows: 1

\* - Code

- Description

- Exemple d'usage

\* - 200

- Succès

- Requête traitée avec succès

\* - 201

- Créé

- Ressource créée avec succès

\* - 400

- Requête invalide

- Données manquantes ou format incorrect

\* - 401

- Non autorisé

- Token manquant ou invalide

\* - 403

- Interdit

- Permissions insuffisantes

\* - 404

- Non trouvé

- Ressource inexistante

\* - 409

- Conflit

- Ressource déjà existante

\* - 422

- Entité non traitable

- Erreurs de validation

\* - 500

- Erreur serveur

- Erreur interne du serveur

\*\*Format des erreurs\*\*

{  
 "error": {  
 "code": "VALIDATION\_ERROR",  
 "message": "Les données fournies sont invalides",  
 "details": [  
 {  
 "field": "email",  
 "message": "Format d'email invalide"  
 }  
 ]  
 }  
}

Limitation et quotas

- \*\*Taille maximale du payload\*\* : 10 MB

\*\*Rate limiting\*\* : Implémenté avec en-têtes X-RateLimit-\*

Webhooks

Configuration des webhooks pour les événements temps réel.

Événements supportés

- ``consultant.updated`` : Consultant modifié

- ``mission.started`` : Nouvelle mission démarrée

Configuration webhook

POST /api/v1/webhooks HTTP/1.1  
Content-Type: application/json

{

"url": "https://mon-app.com/webhooks/consultator",

"events": ["consultant.created", "consultant.updated"],

"secret": "mon-secret-webhook"

}

Format des payloads webhook

{  
 "event": "consultant.created",  
 "timestamp": "2024-01-15T10:30:00Z",  
 "data": {  
 "consultant": {  
 "id": 123,  
 "nom": "DUPONT",  
 "prenom": "Jean"  
 }  
 },  
 "signature": "sha256=abc123..."  
}

SDKs et bibliothèques

\*\*Python SDK\*\*

from consultator\_sdk import Client

client = Client(api\_key="your-api-key")

# Lister les consultants

consultants = client.consultants.list(search="Python")

# Créer un consultant

consultant = client.consultants.create({

"nom": "MARTIN",

"prenom": "Sophie",

"competences": [{"nom": "Data Science", "annees\_experience": 4}]

})

\*\*JavaScript SDK\*\*

import { Consultator } from 'consultator-sdk';

const client = new Consultator({ apiKey: 'your-api-key' });

// Récupérer un consultant

const consultant = await client.consultants.get(123);

// Mettre à jour un consultant

await client.consultants.update(123, {

email: 'nouveau.email@consultant.com'

});

Migration depuis l'API actuelle

Pour migrer depuis l'utilisation directe des services Python :

\*\*Remplacer les imports de services\*\*

.. code-block:: python

# Avant

from app.services.consultant\_service import ConsultantService

consultants = ConsultantService.get\_all\_consultants()

# Après

import requests

response = requests.get('/api/v1/consultants',

headers={'Authorization': 'Bearer ' + token})

consultants = response.json()['data']

\*\*Adapter la gestion d'erreurs\*\*

.. code-block:: python

# Gestion d'erreurs HTTP

try:

response = requests.post('/api/v1/consultants', json=data)

response.raise\_for\_status()

consultant = response.json()

except requests.exceptions.HTTPError as e:

if e.response.status\_code == 422:

errors = e.response.json()['error']['details']

# Traiter les erreurs de validation

\*\*Implémenter la pagination\*\*

.. code-block:: python

def get\_all\_consultants(token, page=1):

consultants = []

while True:

response = requests.get(f'/api/v1/consultants?page={page}',

headers={'Authorization': f'Bearer {token}'})

data = response.json()

consultants.extend(data['data'])

if page >= data['pagination']['pages']:

break

page += 1

return consultants

Support et évolution

L'API évolue selon le principe de compatibilité ascendante. Les changements breaking seront annoncés 3 mois à l'avance.

\*\*Versions supportées\*\*

\*\*v2\*\* (future) : Nouvelles fonctionnalités (Q2 2025)

\*\*Support\*\*

- \*\*Status page\*\* : `https://status.consultator.com <https://status.consultator.com>`\_

Tutoriels

Cette section contient des guides pratiques pour maîtriser les fonctionnalités avancées de Consultator.

Import de données Excel

Apprenez à importer efficacement vos données de consultants depuis Excel.

Préparation des données

\*\*Format requis\*\*

Votre fichier Excel doit contenir les colonnes suivantes :

.. list-table:: Colonnes obligatoires

:header-rows: 1

\* - Colonne

- Type

- Description

- Exemple

\* - nom

- Texte

- Nom de famille en majuscules

- DUPONT

\* - prenom

- Texte

- Prénom

- Jean

\* - email

- Texte

- Adresse email professionnelle

- jean.dupont@consultant.com

.. list-table:: Colonnes optionnelles

:header-rows: 1

\* - Colonne

- Type

- Description

\* - telephone

- Texte

- Numéro de téléphone

\* - competence\_1

- Texte

- Première compétence

\* - annees\_exp\_1

- Nombre

- Années d'expérience pour competence\_1

\* - competence\_2

- Texte

- Deuxième compétence

\* - annees\_exp\_2

- Nombre

- Années d'expérience pour competence\_2

\*\*Exemple de fichier Excel\*\*

:header: nom, prenom, email, telephone, competence\_1, annees\_exp\_1, competence\_2, annees\_exp\_2

DUPONT,Jean,jean.dupont@email.com,+33123456789,Python,5,SQL,3

MARTIN,Marie,marie.martin@email.com,+33987654321,Data Science,4,Machine Learning,2

BERNARD,Paul,paul.bernard@email.com,+33555666777,Java,8,Spring,6

Import étape par étape

\*\*Accès à la fonctionnalité d'import\*\*

- Lancez Consultator

- Allez dans "👥 Consultants"

- Cliquez sur "📥 Importer des données"

\*\*Sélection du fichier\*\*

- Cliquez sur "Choisir un fichier"

- Sélectionnez votre fichier Excel (.xlsx ou .xls)

- Vérifiez que le fichier est correctement chargé

\*\*Configuration du mapping\*\*

.. image:: \_static/import\_mapping.png

:alt: Interface de mapping des colonnes

- Vérifiez automatiquement le mapping des colonnes

- Ajustez manuellement si nécessaire

- Consultez l'aperçu des données

4. \*\*Validation des données\*\*

Le système vérifie automatiquement :

- \*\*Format des emails\*\* : Validation des adresses email

- \*\*Cohérence des compétences\*\* : Vérification des noms de compétences

- \*\*Doublons\*\* : Détection des consultants déjà existants

5. \*\*Import final\*\*

- Lancez l'import

- Suivez la progression en temps réel

- Consultez le rapport d'import

Gestion des erreurs d'import

\*\*Erreurs courantes et solutions\*\*

.. list-table:: Problèmes fréquents

:header-rows: 1

\* - Erreur

- Cause

- Solution

\* - Email invalide

- Format d'email incorrect

- Vérifier et corriger l'adresse email

\* - Compétence inconnue

- Compétence non référencée

- Ajouter la compétence au référentiel ou corriger le nom

\* - Doublon détecté

- Consultant déjà existant

- Choisir de mettre à jour ou ignorer

\* - Colonne manquante

- Colonne obligatoire absente

- Ajouter la colonne ou modifier le mapping

\*\*Rapport d'erreurs détaillé\*\*

Après l'import, consultez le rapport qui détaille :

- Liste des erreurs par ligne

Création de rapports personnalisés

Apprenez à créer des rapports sur mesure pour vos besoins d'analyse.

Types de rapports disponibles

\*\*Rapports standards\*\*

- \*\*Rapport par compétence\*\* : Analyse détaillée des compétences

\*\*Rapport RH\*\* : Évolution des effectifs et turnover

\*\*Rapports personnalisés\*\*

Création d'un rapport sur mesure :

\*\*Définition des critères\*\*

- Période d'analyse (date de début/fin)

- Filtres sur les consultants (compétences, statut, etc.)

- Métriques à inclure

- Format de sortie (PDF, Excel, HTML)

\*\*Configuration des métriques\*\*

.. code-block:: python

# Exemple de configuration de rapport

report\_config = {

"title": "Rapport Data Science Q4 2024",

"period": {

"start": "2024-10-01",

"end": "2024-12-31"

},

"filters": {

"skills": ["Python", "Machine Learning", "Data Science"],

"status": "active"

},

"metrics": [

"consultant\_count",

"average\_revenue",

"skill\_distribution",

"project\_completion\_rate"

],

"format": "pdf"

}

\*\*Génération automatique\*\*

- Planification des rapports récurrents

- Export automatique vers le cloud

- Distribution par email

Analyse des compétences

\*\*Matrice de compétences\*\*

Visualisez les compétences de votre équipe :

# Génération d'une matrice compétence/niveau  
skill\_matrix = consultant\_service.generate\_skill\_matrix(  
 consultants=active\_consultants,  
 skills=["Python", "SQL", "Machine Learning", "AWS"],  
 group\_by="seniority"  
)

\*\*Gap analysis\*\*

Identifiez les besoins en recrutement :

# Analyse des écarts de compétences  
gaps = analytics\_service.identify\_skill\_gaps(  
 required\_skills=project\_requirements,  
 available\_skills=current\_team\_skills,  
 timeframe\_months=6  
)

\*\*Prévisions de compétences\*\*

# Prévision des besoins futurs  
forecast = analytics\_service.forecast\_skill\_demand(  
 historical\_data=mission\_history,  
 market\_trends=market\_data,  
 prediction\_months=12  
)

Configuration du chatbot IA

Configurez et optimisez le chatbot IA pour vos besoins métier.

Configuration de base

\*\*Accès aux paramètres\*\*

- Allez dans "⚙️ Configuration"

- Section "🤖 Chatbot IA"

\*\*Configuration des sources de connaissances\*\*

- \*\*Base de données consultants\*\* : Activé par défaut

- \*\*Documents uploadés\*\* : CV, présentations, etc.

- \*\*Base de connaissances métier\*\* : Documents internes

\*\*Paramètres de réponse\*\*

- \*\*Style de réponse\*\* : Professionnel, Conversationnel, Technique

- \*\*Niveau de détail\*\* : Synthétique, Détaillé, Exhaustif

- \*\*Langue\*\* : Français, Anglais

Entraînement du modèle

\*\*Ajout de connaissances spécifiques\*\*

# Entraînement sur des documents métier  
chatbot\_service.train\_on\_documents([  
 "docs/referentiel\_competences.pdf",  
 "docs/methodologie\_projets.docx",  
 "data/base\_connaissances.json"  
])

\*\*Personnalisation des réponses\*\*

# Configuration des réponses personnalisées  
custom\_responses = {  
 "tarifs": "Nos tarifs varient de 500€ à 1200€ par jour selon l'expertise.",  
 "delais": "Les délais de mobilisation sont généralement de 2-4 semaines.",  
 "methodologie": "Nous suivons une approche agile avec Scrum/Kanban."  
}

chatbot\_service.add\_custom\_responses(custom\_responses)

Utilisation avancée

\*\*Requêtes complexes\*\*

Le chatbot peut traiter des demandes sophistiquées :

- \*"Peux-tu me proposer une équipe de 3 personnes pour un projet Data Science de 6 mois ?"\*

\*\*Intégration avec les analyses\*\*

# Génération de rapports via le chatbot  
report\_request = "Génère-moi un rapport des compétences Data Science pour le trimestre dernier"  
report = chatbot\_service.generate\_report\_from\_query(report\_request)

# Analyse prédictive

prediction = chatbot\_service.predict\_from\_query(

"Quelles compétences seront demandées dans 6 mois ?"

)

Maintenance et optimisation

\*\*Mise à jour des connaissances\*\*

# Mise à jour périodique des connaissances  
python scripts/update\_chatbot\_knowledge.py

\*\*Monitoring des performances\*\*

- Temps de réponse moyen

Améliorations suggérées

Intégration API externe

Connectez Consultator à vos outils existants via l'API.

Authentification API

\*\*Obtention d'un token\*\*

curl -X POST https://api.consultator.com/v1/auth/login \  
 -H "Content-Type: application/json" \  
 -d '{"username": "votre-email", "password": "votre-mot-de-passe"}'

\*\*Utilisation du token\*\*

curl -H "Authorization: Bearer VOTRE\_TOKEN" \  
 https://api.consultator.com/v1/consultants

Synchronisation avec un CRM

\*\*Export vers Salesforce\*\*

import requests  
from simple\_salesforce import Salesforce

def sync\_consultants\_to\_salesforce():

# Récupération des consultants

response = requests.get(

"https://api.consultator.com/v1/consultants",

headers={"Authorization": f"Bearer {token}"}

)

consultants = response.json()["data"]

# Connexion Salesforce

sf = Salesforce(username=username, password=password, security\_token=token)

# Synchronisation

for consultant in consultants:

sf.Contact.create({

"LastName": consultant["nom"],

"FirstName": consultant["prenom"],

"Email": consultant["email"],

"Title": "Consultant"

})

\*\*Import depuis HubSpot\*\*

from hubspot import HubSpot

def import\_contacts\_from\_hubspot():

api\_client = HubSpot(access\_token=access\_token)

contacts = api\_client.crm.contacts.get\_all()

for contact in contacts:

consultant\_data = {

"nom": contact.properties["lastname"],

"prenom": contact.properties["firstname"],

"email": contact.properties["email"]

}

requests.post(

"https://api.consultator.com/v1/consultants",

json=consultant\_data,

headers={"Authorization": f"Bearer {token}"}

)

Intégration avec Jira

\*\*Synchronisation des projets\*\*

from jira import JIRA

def sync\_projects\_from\_jira():

jira = JIRA(server=jira\_url, basic\_auth=(username, password))

# Récupération des projets actifs

projects = jira.projects()

for project in projects:

# Création d'une mission dans Consultator

mission\_data = {

"nom\_projet": project.name,

"client": project.lead.displayName,

"statut": "en\_cours"

}

requests.post(

"https://api.consultator.com/v1/missions",

json=mission\_data,

headers={"Authorization": f"Bearer {token}"}

)

Webhook pour les événements

\*\*Configuration d'un webhook\*\*

webhook\_config = {  
 "url": "https://mon-app.com/webhooks/consultator",  
 "events": ["consultant.created", "mission.completed"],  
 "secret": "mon-secret-webhook"  
}

requests.post(

"https://api.consultator.com/v1/webhooks",

json=webhook\_config,

headers={"Authorization": f"Bearer {token}"}

)

\*\*Traitement des événements\*\*

from flask import Flask, request  
import hmac  
import hashlib

app = Flask(\_\_name\_\_)

@app.route('/webhooks/consultator', methods=['POST'])

def handle\_webhook():

payload = request.get\_data()

signature = request.headers.get('X-Consultator-Signature')

# Vérification de la signature

expected\_signature = hmac.new(

webhook\_secret.encode(),

payload,

hashlib.sha256

).hexdigest()

if not hmac.compare\_digest(signature, expected\_signature):

return 'Invalid signature', 400

# Traitement de l'événement

event\_data = request.get\_json()

event\_type = event\_data['event']

if event\_type == 'consultant.created':

# Mise à jour du CRM externe

update\_external\_crm(event\_data['data']['consultant'])

return 'OK', 200

Optimisation des performances

Techniques pour maintenir des performances optimales avec une base de données importante.

Indexation de la base de données

\*\*Index stratégiques\*\*

-- Index sur les recherches fréquentes  
CREATE INDEX idx\_consultant\_email ON consultants(email);  
CREATE INDEX idx\_consultant\_nom\_prenom ON consultants(nom, prenom);

-- Index sur les compétences

CREATE INDEX idx\_competence\_nom ON competences(nom);

CREATE INDEX idx\_consultant\_competence\_consultant\_id ON consultant\_competence(consultant\_id);

CREATE INDEX idx\_consultant\_competence\_competence\_id ON consultant\_competence(competence\_id);

-- Index composites pour les analyses

CREATE INDEX idx\_mission\_consultant\_dates ON missions(consultant\_id, date\_debut, date\_fin);

\*\*Maintenance des index\*\*

# Analyse et reconstruction des index  
python scripts/optimize\_database\_indexes.py

Cache intelligent

\*\*Cache des requêtes fréquentes\*\*

from streamlit import cache\_data  
import time

@cache\_data(ttl=3600) # Cache pendant 1 heure

def get\_consultant\_stats():

"""Cache les statistiques des consultants."""

return consultant\_service.get\_statistics()

@cache\_data(ttl=1800) # Cache pendant 30 minutes

def get\_skill\_distribution():

"""Cache la distribution des compétences."""

return analytics\_service.get\_skill\_distribution()

\*\*Cache des données de référence\*\*

@st.cache\_data  
def load\_referentiel\_competences():  
 """Cache le référentiel des compétences."""  
 return competence\_service.get\_all\_competences()

Optimisation des requêtes

\*\*Requêtes paginées\*\*

def get\_consultants\_optimized(page=1, per\_page=50, filters=None):  
 """Récupération optimisée avec pagination."""  
 query = session.query(Consultant)

# Application des filtres

if filters:

if filters.get('skill'):

query = query.join(Consultant.competences).filter(

Competence.nom.ilike(f"%{filters['skill']}%")

)

if filters.get('status'):

query = query.filter(Consultant.statut == filters['status'])

# Pagination

offset = (page - 1) \* per\_page

return query.offset(offset).limit(per\_page).all()

\*\*Chargement sélectif\*\*

def get\_consultant\_summary(consultant\_id):  
 """Chargement sélectif des données essentielles."""  
 return session.query(  
 Consultant.id,  
 Consultant.nom,  
 Consultant.prenom,  
 Consultant.email,  
 func.count(Competence.id).label('nb\_competences')  
 ).join(Consultant.competences).filter(  
 Consultant.id == consultant\_id  
 ).group\_by(Consultant.id).first()

Monitoring des performances

\*\*Métriques clés à surveiller\*\*

- \*\*Temps de chargement des données\*\* : < 5 secondes pour 1000 consultants

\*\*Temps de requête base de données\*\* : < 500ms en moyenne

\*\*Outils de monitoring\*\*

import time  
import logging

def monitor\_query\_performance(func):

"""Décorateur pour mesurer les performances des requêtes."""

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

execution\_time = time.time() - start\_time

if execution\_time > 1.0: # Log si > 1 seconde

logging.warning(f"Requête lente: {func.\_\_name\_\_} - {execution\_time:.2f}s")

return result

return wrapper

@monitor\_query\_performance

def get\_consultants\_with\_skills():

return consultant\_service.get\_all\_with\_competences()

Maintenance préventive

\*\*Tâches automatisées\*\*

# Optimisation quotidienne de la base  
0 2 \* \* \* python scripts/daily\_database\_maintenance.py

# Reconstruction des index hebdomadaire

0 3 \* \* 1 python scripts/weekly\_index\_rebuild.py

# Archive des données anciennes mensuelle

0 4 1 \* \* python scripts/monthly\_data\_archive.py

\*\*Seuils d'alerte\*\*

- Taille base de données > 10GB : archivage recommandé