🤝 Guide de Contribution - Chatbot Desktop

Merci de votre intérêt pour contribuer à Chatbot Desktop ! Ce guide vous aidera à démarrer.

📋 Table des Matières

🤗 Code de Conduite

Notre Engagement

Nous nous engageons à faire de la participation à ce projet une expérience exempte de harcèlement pour tous, indépendamment de :

- L'âge, la taille corporelle, le handicap
- L'origine ethnique, l'identité de genre
- Le niveau d'expérience, la nationalité
- L'apparence personnelle, la race, la religion
- L'identité ou l'orientation sexuelle

Nos Standards

Comportements encouragés :

- ✅ Utiliser un langage accueillant et inclusif
- ✅ Respecter les différents points de vue
- ✅ Accepter les critiques constructives avec grâce
- ✅ Se concentrer sur ce qui est meilleur pour la communauté
- ✅ Faire preuve d'empathie envers les autres

Comportements inacceptables :

- ❌ Langage ou images à caractère sexuel
- ❌ Trolling, commentaires insultants/dérogatoires

- ❌ Harcèlement public ou privé
- ❌ Publication d'informations privées d'autrui
- ❌ Toute conduite non professionnelle

🚀 Comment Contribuer

Signaler des Bugs

Les bugs sont suivis via les **Issues GitHub**. Avant de créer un rapport :

1. **Vérifier** qu'il n'existe pas déjà
2. **Utiliser** le template de bug report
3. **Inclure** :
   - Description claire du problème
   - Étapes pour reproduire
   - Comportement attendu vs réel
   - Logs en mode `--debug`
   - Version Python et OS
   - Captures d'écran si pertinent

Proposer des Fonctionnalités

1. **Ouvrir une Issue** avec le label `enhancement`
2. **Décrire** :
   - Cas d'usage précis
   - Bénéfice pour les utilisateurs
   - Implémentation suggérée (optionnel)
3. **Discuter** avec les mainteneurs

Contribuer du Code

1. Fork & Clone

```bash
# Fork sur GitHub, puis :
git clone https://github.com/votre-username/chatbot-desktop.git
cd chatbot-desktop
```

2. Créer une Branche

```bash
git checkout -b feature/ma-nouvelle-fonctionnalite
# ou
git checkout -b fix/correction-bug-xyz
```

Convention de nommage :

- `feature/` : Nouvelle fonctionnalité
- `fix/` : Correction de bug
- `docs/` : Documentation
- `refactor/` : Refactoring
- `test/` : Ajout de tests

3. Développer

```bash
# Installer en mode développement
python -m venv venv
source venv/bin/activate  # ou venv\Scripts\activate sur Windows
pip install -r requirements.txt
```

4. Tester

```bash
# Tests manuels
python main.py --debug

# Vérifier que tout fonctionne :
# - Création conversation
# - Envoi messages
# - Export JSON/MD
# - Paramètres
```

5. Commit

```bash
git add .
git commit -m "feat: Ajoute support des images"
```

Convention de commit (Conventional Commits) :

- `feat:` Nouvelle fonctionnalité
- `fix:` Correction de bug
- `docs:` Documentation
- `style:` Formatage (pas de changement de code)
- `refactor:` Refactoring
- `test:` Ajout de tests
- `chore:` Maintenance

6. Push & Pull Request

```bash
git push origin feature/ma-nouvelle-fonctionnalite
```

Puis créer une Pull Request sur GitHub.

🏗️ Architecture du Projet

Structure

```
chatbot_desktop/
├── core/          # Business logic
├── ui/            # Interface PyQt6
├── workers/       # Threads asynchrones
└── utils/         # Utilitaires
```

Principes

1. **Séparation MVC**
   - Models : `core/database.py`
   - Views : `ui/*.py`
   - Controller : `core/main_controller.py`
2. **Communication Asynchrone**
   - Utiliser signaux/slots PyQt6
   - Workers pour opérations longues
3. **Logging Systématique**
   - Toutes les opérations importantes
   - Mode DEBUG pour développement

4. **Gestion d'Erreurs**
    - Try/catch avec logging
    - Messages utilisateur clairs

📐 Standards de Code

Python

Style : PEP 8

```python
# Bon
def calculate_sum(numbers: List[int]) -> int:
    """Calcule la somme d'une liste de nombres."""
    return sum(numbers)

# Mauvais
def calc(n):
    return sum(n)
```

Type Hints : Obligatoires

```python
from typing import List, Dict, Optional

def get_conversation(conv_id: int) -> Optional[Dict]:
    """Récupère une conversation."""
    pass
```

Docstrings : Format Google

```python
```

```python
def my_function(param1: str, param2: int) -> bool:
    """
    Description courte.

    Description longue si nécessaire.

    Args:
        param1: Description du paramètre
        param2: Description du paramètre

    Returns:
        Description du retour

    Raises:
        ValueError: Si param2 est négatif
    """
    pass
```

Imports : Organisés

```python
# Standard library
import sys
from datetime import datetime

# Third party
from PyQt6.QtWidgets import QWidget
from PyQt6.QtCore import pyqtSignal

# Local
from core.logger import get_logger
from utils.html_generator import HTMLGenerator
```

PyQt6

Signaux : Nommage clair

```python
```

```
# Bon
conversation_selected = pyqtSignal(int)
message_submitted = pyqtSignal(str)

# Mauvais
sig1 = pyqtSignal(int)
msg = pyqtSignal(str)
```

Layout : Toujours utiliser des layouts

```
python

# Bon
layout = QVBoxLayout(self)
layout.addWidget(widget)

# Mauvais
widget.move(10, 20)  # Positionnement absolu
```

🧪 Tests

Tests Manuels Obligatoires

Avant chaque PR, vérifier :

- [ ] ✅ Lancement normal : `python main.py`
- [ ] ✅ Lancement debug : `python main.py --debug`
- [ ] ✅ Création de conversation
- [ ] ✅ Envoi de message avec streaming
- [ ] ✅ Sélection multiple (Shift+Clic)
- [ ] ✅ Suppression de conversations
- [ ] ✅ Export JSON et Markdown
- [ ] ✅ Paramètres (test connexion)
- [ ] ✅ Personnalisation couleurs
- [ ] ✅ Raccourcis clavier

Logs

Vérifier qu'aucun WARNING/ERROR en mode debug.

📝 Processus de Pull Request

Checklist avant Soumission

- [ ] Code suit les standards PEP 8
- [ ] Type hints ajoutés

- [ ] Docstrings présentes
- [ ] Logs ajoutés si pertinent
- [ ] Tests manuels passés
- [ ] Commit messages conformes
- [ ] Branche à jour avec `main`

Template de PR

markdown

```
## Description

[Description claire des changements]

## Type de Changement

- [ ] 🐛 Bug fix
- [ ] ✨ Nouvelle fonctionnalité
- [ ] 📝 Documentation
- [ ] 🎨 Style/UI
- [ ] ♻️ Refactoring

## Tests Effectués

- [ ] Test 1
- [ ] Test 2

## Captures d'Écran

[Si pertinent]

## Notes Additionnelles

[Informations supplémentaires]
```

Revue de Code

Les mainteneurs vont :

1. Vérifier le code
2. Tester manuellement
3. Proposer des améliorations si besoin
4. Approuver ou demander des modifications

Délai : Généralement 2-5 jours ouvrés

🎯 Domaines de Contribution

Facile (Good First Issue)

- 📝 Documentation
- 🐛 Petits bugs
- 🎨 Améliorations UI mineures

Moyen

- ✨ Nouvelles fonctionnalités simples
- ♻️ Refactoring
- 🧪 Ajout de tests

Difficile

- 🏗️ Changements architecturaux
- 🚀 Optimisations performances
- 🔒 Sécurité

💬 Communication

- **Issues** : Bugs et features
- **Pull Requests** : Code reviews
- **Discussions** : Questions générales

🙏 Remerciements

Merci à tous les contributeurs qui rendent ce projet meilleur !

Questions ? N'hésitez pas à ouvrir une Discussion GitHub.