Documentation Todoux

Vue d'ensemble du Projet

Introduction

Todoux est une application de gestion de tâches collaborative développée en Python. Elle permet aux utilisateurs de créer, gérer et partager des tâches au sein de groupes et de dossiers, avec une interface utilisateur moderne et intuitive.

Fonctionnalités Principales

- Gestion complète des tâches (CRUD)
- Collaboration en temps réel
- Intégration avec Google Calendar
- Système de permissions avancé
- Interface utilisateur moderne avec thèmes
- Assistance vocale (AI VOC)
- Import/Export de données

Structure du Projet

Organisation des Dossiers

- **API/** : Contient les endpoints de l'API REST
- **scripts_google/** : Scripts d'intégration avec Google Calendar
- **test_unitaire/** : Tests unitaires de l'application
- **help_sections/** : Documentation utilisateur en HTML
- **img/** : Ressources graphiques
- **replication/** : Gestion de la synchronisation des données
- **videos/** : Tutoriels vidéo

Fichiers Principaux

- **Application.py** : Cœur de l'application avec l'interface utilisateur
- **installation.sh** : Script d'installation automatisée
- **Base_De_Donnée** : Schéma et scripts SQL
- **help.html** : Documentation utilisateur

Architecture Technique

Stack Technologique

- **Backend** : Python 3.8+
- **Base de données** : MySQL 5.7+
- **Interface graphique** : PyQt6
- **API** : Flask avec Gunicorn

- **Conteneurisation** : Docker et Docker Compose
- **Authentification** : JWT + 2FA

API REST

TodoList API est une solution complète de gestion de tâches collaborative offrant :

- Gestion hiérarchique des tâches (Groupes > Dossiers > Tâches)
- Système de permissions avancé
- Intégration avec Google Calendar
- Historique complet des actions
- API RESTful sécurisée

1. *Composants principaux*
   Frontend Proxy (Apache)
   - *Gestion SSL/TLS*
   - *Reverse proxy*
   - *Compression gzip*
   - *Cache statique*

2. *Application Server (Gunicorn)*
   - *3 workers*
   - *2 threads par worker*
   - *Timeout de 120s*
   - *Keepalive de 5s*

3. *API Layer (Flask)*
   - *Routing*
   - *Middleware*
   - *Validation*
   - *Sérialisation*

4. *Database (MySQL)*
   - *Tables relationnelles*
   - *Indexation optimisée*
   - *Contraintes d'intégrité*
   - *Backup automatisé*

Pour toute informations supplémentaires sur l'API veuillez suivre ce lien :
https://github.com/jameschm/R504---To-Do-List/blob/main/docs/documentation_api.md

Infrastructure Docker

L'application est conteneurisée avec Docker pour faciliter le déploiement :

Conteneurs

- **app** : Application PyQt6
- **db** : Base de données MySQL

Configuration

- Utilisation de Docker Compose
- Volumes pour la persistance des données
- Réseau isolé entre les conteneurs
- Variables d'environnement pour la configuration

Déploiement

- Scripts de déploiement automatisé
- Gestion des environnements (dev/prod)
- Sauvegarde automatique des données
- Monitoring des conteneurs

Pour Docker, nous avons également créé un fichier .desktop afin d'intégrer une icône, facilitant ainsi le lancement de l'application sans devoir passer par la ligne de commande. Cette solution permet d'avoir une application "professionnelle" avec une interface utilisateur conviviale.

Dépendances Système

L'application nécessite plusieurs dépendances système : - Qt5 (GUI) - Bibliothèques X11 - MySQL Client - Dépendances pour le rendu graphique

Réplication et Synchronisation

Architecture de Réplication

- **api_client.py** : Client API pour la synchronisation
- **config.py** : Configuration de la réplication
- **db_sync.py** : Synchronisation de la base de données
- **sync_service.py** : Service de synchronisation

Fonctionnalités de Synchronisation

- Réplication bidirectionnelle
- Gestion des conflits
- Synchronisation différentielle
- Vérification d'intégrité

Configuration de la Réplication

```
SYNC_CONFIG = {
    'interval': 300,  # 5 minutes
    'retry_delay': 60,
    'max_retries': 3,
```

```
    'batch_size': 100
}
```

Modes de Synchronisation

- Synchronisation automatique
- Synchronisation manuelle
- Synchronisation à la demande
- Synchronisation différée

Structure de la Base de Données

Schéma Physique

Schéma de la base de données

Base de Données

La base de données MySQL comprend les tables suivantes :

- USER : Gestion des utilisateurs.
- GROUPE : Gestion des groupes collaboratifs.
- DOSSIER : Organisation des tâches en catégories.
- TACHES : Gestion des tâches principales.
- DROIT : Contrôle des permissions sur les tâches.
- ETIQUETTES : Système de tags.
- TACHE_ETIQUETTE : Association entre tâches et étiquettes.
- INVITATION : Gestion des invitations à des groupes.
- MEMBRE : Suivi des membres des groupes.
- HISTORIQUE : Suivi des actions sur les tâches.
- SOUS_TACHES : Gestion des sous-tâches.
- COMMENTAIRES : Stockage des commentaires sur les tâches.

Relations

- USER_GROUPE : Association entre utilisateurs et groupes.
- TACHE_USER : Attribution des tâches aux utilisateurs.
- TACHE_ETIQUETTE : Étiquetage des tâches avec des tags.
- GROUPE_INVITATION : Association entre les groupes et les invitations.
- GROUPE_MEMBRE : Gestion des relations entre groupes et membres.
- TACHE_HISTORIQUE : Enregistrement des actions réalisées sur les tâches.
- TACHE_SOUS_TACHE : Relation entre les tâches principales et leurs sous-tâches.
- TACHE_COMMENTAIRES : Association des tâches avec leurs commentaires.

Indexation

- Index sur les clés étrangères

- Index de recherche full-text
- Index composites pour les requêtes fréquentes

Tests et Qualité du Code

Tests Unitaires

L'application dispose d'une suite complète de tests unitaires :

1. **Tests Fonctionnels** :
   - Fonctions utilitaires
   - Interactions base de données
   - Composants interface utilisateur
   - Interactions spécifiques
   - Gestion des utilisateurs
   - Fonctionnalités de l'application
   - Thèmes et styles
2. **Tests Métier** :
   - Gestion des sous-tâches
   - Gestion des commentaires
   - Sélection de groupe
   - Gestion des utilisateurs
   - Gestion des dossiers
   - Gestion des tâches
   - Gestion des membres
   - Gestion des fenêtres
   - Gestion des étiquettes

Tests API

- Tests des endpoints
- Tests de sécurité
- Tests d'authentification
- Tests de charge

Intégration Google Calendar

Scripts d'Intégration

- **cal_setup.py** : Configuration de l'authentification Google
- **calendars_get.py** : Récupération des calendriers
- **create_calendars.py** : Création de nouveaux calendriers
- **create_events.py** : Création d'événements
- **events_get.py** : Récupération des événements
- **script_google_export.py** : Export vers Google Calendar
- **script_google_import.py** : Import depuis Google Calendar

Configuration

- Authentification OAuth2
- Gestion des credentials
- Gestion des tokens
- Permissions requises

Fonctionnalités

- Synchronisation bidirectionnelle
- Gestion des conflits
- Mise à jour automatique
- Gestion des erreurs

Choix d'Implémentation

Interface Utilisateur (PyQt6)

Raisons du choix : - Framework mature et stable - Support natif du multi-plateformes - Performances optimales - Riche en widgets - Support des thèmes

Gestion des Données (MySQL)

Avantages : - Robustesse et fiabilité - Support des transactions ACID - Excellentes performances - Large communauté - Outils d'administration matures

Sécurité

- Authentification à deux facteurs
- Hachage des mots de passe avec bcrypt
- Validation des entrées utilisateur
- Protection contre les injections SQL
- Sessions sécurisées

Fonctionnalités Avancées

Intégration Google Calendar

- Synchronisation bidirectionnelle
- Gestion des conflits
- Import/Export sélectif
- Mise à jour automatique

Import/Export de Données

Base de Données

- **Export de la base de données** :
  - Sauvegarde complète de la structure et des données
  - Possibilité d'export sélectif par table
  - Export au format SQL ou CSV

    - Compression automatique des sauvegardes
- **Import de la base de données** :
    - Restauration complète ou partielle
    - Validation des données importées
    - Gestion des conflits de clés
    - Support des jeux de caractères

Formats d'Export

- **SQL** : Format natif pour les sauvegardes complètes
- **CSV** : Pour l'import/export de données tabulaires
- **Excel** : Pour une meilleure lisibilité et édition
- **JSON** : Pour l'interopérabilité avec d'autres systèmes

Automatisation

- Scripts de backup automatiques
- Rotation des sauvegardes
- Notifications en cas d'échec
- Vérification de l'intégrité des données

Assistance Vocale (AI VOC)

- Reconnaissance vocale
- Synthèse vocale
- Commandes personnalisables
- Support multilingue

Système de Filtres

- Filtrage par statut
- Filtrage par étiquettes
- Recherche full-text
- Tri personnalisable

Limitations et Perspectives

Limitations Actuelles

- Performance avec grands volumes de données
- Pas de support hors ligne
- Interface glisser-déposer limitée
- Pas de notifications push

Améliorations Futures

- Mode hors ligne
- Application mobile
- API publique
- Intégration d'autres calendriers

- Système de plugins

Installation et Déploiement

Déploiement avec Docker

1. Prérequis :
   - Docker
   - Docker Compose
   - Git
2. Installation :

```
git clone [repository]
cd todoux
docker-compose up -d
```

3. Configuration :
   - Copier `.env.example` vers `.env`
   - Configurer les variables d'environnement
   - Initialiser la base de données :

```
docker-compose exec api flask db upgrade
```

Installation Manuelle

- Prérequis :
  - Python 3.8+
  - MySQL 5.7+
  - Qt5
  - Docker (optionnel)
- Procédure d'Installation L'installation peut se faire de deux manières :
  1. Via le script `installation.sh` (Linux/macOS)
  2. Manuellement en suivant la documentation
- Configuration
  - Configuration de la base de données
  - Paramètres de l'application
  - Intégration Google Calendar
  - Configuration du serveur API

Maintenance

Tests

- Tests unitaires avec unittest
- Tests d'intégration

- Tests de l'interface utilisateur

Bonnes Pratiques

- Documentation du code
- Gestion des versions
- Revue de code
- Tests automatisés

Recommandations

- Mises à jour régulières
- Sauvegardes de la base de données
- Monitoring des performances
- Formation des utilisateurs

Thèmes et Personnalisation

Thèmes Disponibles

- Thème clair
- Thème sombre
- Thèmes personnalisés

Styles CSS

```
QWidget {
    background-color: #2c3e50;
    color: #ecf0f1;
    font-family: 'Arial', sans-serif;
}

QLabel#title_label {
    font-size: 36px;
    font-weight: bold;
    color: #ecf0f1;
}
```

Personnalisation

- Couleurs personnalisables
- Polices configurables
- Icônes modifiables
- Layouts adaptables

Sécurité

Authentification

- Système 2FA
- Hachage des mots de passe (bcrypt)
- Gestion des sessions

- Tokens JWT

Protection des Données

- Chiffrement des données sensibles
- Validation des entrées
- Protection XSS
- Protection CSRF
- Rate limiting

Journalisation

- Logs d'accès
- Logs d'erreurs
- Logs d'audit
- Rotation des logs

Déploiement

Environnements

- Développement
- Test
- Production

Configuration par Environnement

```
# Développement
DEBUG=True
DB_HOST=localhost
API_URL=http://localhost:5000

# Production
DEBUG=False
DB_HOST=db.production
API_URL=https://todoux.com
```

Procédures de Déploiement

1. **Développement** :

   ```
   docker-compose -f docker-compose.dev.yml up
   ```

2. **Production** :

   ```
   docker-compose -f docker-compose.prod.yml up -d
   ```

Monitoring

- Surveillance des logs
- Métriques de performance
- Alertes automatiques
- Tableaux de bord