

Miniprojet : Task Mangement python(flask)



ENCADRE PAR:

REALISER PAR:

PR.BAKKALI YEDRI Othman RABIH SENHAJI Anas

29 Mars 2025



Sommaire

1. Introduction
 - 1.1 Contexte du projet
 - 1.2 Objectifs
 - 1.3 Fonctionnalités
 - 1.4 Contenu du rapport
2. Architecture du Projet
 - 2.1 Schéma global du système
 - 2.2 Technologies utilisées
 - 2.3 Structure des répertoires
3. Implémentation Détaillée
 - 3.1 Modèles de données
 - 3.2 Contrôleurs principaux
 - 3.3 Routes clés
 - 3.4 Mécanismes de sécurité
4. Capture d'écran de l'application
 - 4.1 Page d'Accueil
 - 4.2 Écran de Connexion
 - 4.3 Écran d'Inscription
 - 4.4 Tableau de Bord
 - 4.5 Création de Tâche
 - 4.6 Confirmation de Suppression
 - 4.7 Page d'Erreur 404
5. Conclusion



Introduction

1.1 Contexte du projet

Dans le cadre de ce [mini-projet](#), nous avons développé une [application web sécurisée](#) de [gestion des tâches](#) en utilisant [Flask](#), un [micro-framework Python](#).

1.2 Objectifs

L'objectif principal est de permettre aux [utilisateurs d'organiser](#), de [suivre](#) et de [gérer](#) leurs [tâches](#) de manière efficace grâce à une [interface simple](#) et [intuitive](#).

1.3 Fonctionnalités

L'application propose :

- Un [système d'authentification](#) sécurisé, garantissant la [confidentialité](#) des [tâches](#) de chaque utilisateur.
- Des fonctionnalités [CRUD](#) (Create, Read, Update, Delete) permettant la [création](#), la [modification](#), la [suppression](#) et l'[affichage](#) des [tâches](#).
- La possibilité de [trier et filtrer](#) les [tâches](#) selon la [date d'échéance](#) et la [priorité](#).

1.4 Contenu du rapport

Ce [rapport](#) présente :

- [L'architecture](#) de l'application.
- Ses principales [fonctionnalités](#).
- [Capture d'écran](#) de l'application.



















Ce [mini-projet](#) vise à appliquer [les concepts appris en cours](#) et à mettre en pratique le [développement d'une application web sécurisée](#) avec [Flask](#).

Architecture du Projet

L'**architecture** du projet suit le modèle **MVC (Modèle-Vue-Contrôleur)**, ce qui permet une organisation claire et modulaire du code.

2.1 Structure des fichiers

✓ TASK MANAGER

- ✓  app
 - >  __pycache__
 - >  controllers
 - >  forms
 - >  models
 - >  routes
 - >  static
 - >  templates
 -  __init__.py
 -  config.py
 -  extensions.py
- >  flask_session
- >  migrations
-  .env
-  db.sqlite3
-  README.md
-  requirements.txt
-  run.py

1. Architecture MVC :

- Modèles : models/ (interaction avec la DB)
- Vues : templates/ + static/ (présentation)
- Contrôleurs : controllers/ (logique métier)

2. Sécurité :

- .env : Stocke les secrets (clés API, credentials DB)
- flask_session/ : Sessions chiffrées côté serveur
- forms/ : Validation centralisée des entrées

3. Extensibilité :

- migrations/ : Gère l'évolution du schéma DB via Alembic
- extensions.py : Initialisation centralisée des plugins Flask

4. Bonnes pratiques :

- Séparation claire des responsabilités
- Configuration externalisée (config.py + .env)
- Structure modulaire (blueprints dans routes/)

Cette structure suit les standards Flask tout en optimisant :

- La sécurité (validation multi-niveaux)
- La maintenabilité (fichiers par fonctionnalité)
- L'évolutivité (migrations, modularité)

2.2 Technologies utilisées

1. Backend

- **Framework** : Flask (Microframework Python)
 - **Blueprints** pour une architecture modulaire
 - **Jinja2** pour le rendu des templates
 - **Flask-WTF** pour la gestion sécurisée des formulaires
 - **Flask-SQLAlchemy (ORM)** pour l'interaction avec la base de données
 - **Flask-Session** pour la gestion des sessions côté serveur
 - **Flask-Migrate** (Alembic) pour les migrations de la base de données

2. Frontend

- **HTML5 & CSS3** pour la structure et le style
- **Bootstrap 5** pour un design responsive et moderne
- **JavaScript** pour les interactions dynamiques
- **Font Awesome & Bootstrap Icons** pour les icônes

3. Base de Données

- **SQLite** (Développement) / **MySQL** (Production)
- **SQLAlchemy** comme **ORM** pour une abstraction de la base de données

4. Sécurité

- **Werkzeug** pour le hachage sécurisé des mots de passe (generate_password_hash, check_password_hash)
- **CSRF Protection (Flask-WTF)** pour prévenir les attaques Cross-Site Request Forgery
- **Validation** des formulaires côté serveur (**WTForms**)
- **Sessions sécurisées** (Flask-Session avec stockage côté serveur)
- **Cookie Encryption** (Uses Flask's SECRET_KEY)

2.3 Structure des répertoires



Implémentation Détaillée

3.1 Modèles de données

models/user.py

```
from ..extensions import db
from werkzeug.security import generate_password_hash, check_password_hash

class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(128))

    tasks = db.relationship('Task', backref='user', lazy=True)
```



```

tasks = db.relationship('Task', backref='user', lazy=True)

def set_password(self, password):
    self.password_hash = generate_password_hash(password)

def check_password(self, password):
    return check_password_hash(self.password_hash, password)

def __repr__(self):
    return f'<User {self.username}>'

```

Points clés :

- Utilisation de `werkzeug.security` pour le hachage sécurisé
- Relation `SQLAlchemy` pour lier utilisateurs et tâches
- Méthodes utilitaires pour la gestion des mots de passe

3.2 Contrôleurs principaux

controllers/auth_controller.py

```

class AuthController:
    @staticmethod
    def register_user(username, email, password):
        if User.query.filter_by(username=username).first():
            return False, 'Username already exists'
        if User.query.filter_by(email=email).first():
            return False, 'Email already exists'

        new_user = User(username=username, email=email)
        new_user.set_password(password)

        db.session.add(new_user)
        db.session.commit()

        return True, 'User registered successfully'

    @staticmethod
    def login_user(username, password, remember=False):
        user = User.query.filter_by(username=username).first()

        if not user or not user.check_password(password):
            return False, 'Invalid username or password'

        session['user_id'] = user.id
        if remember:
            session.permanent = True

        return True, 'Login successful'

    @staticmethod
    def logout_user():
        session.pop('user_id', None)
        return True, 'Logout successful'

    @staticmethod
    def get_current_user():
        if 'user_id' not in session:
            return None
        return User.query.get(session['user_id'])

```

1. **Hachage sécurisé des mots de passe:**
`set_password(password)` utilise un hachage sécurisé (ex: `bcrypt`) pour ne pas stocker les mots de passe en clair.
2. **Vérification du mot de passe :**
`check_password(password)`
 compare le mot de passe fourni avec celui stocké sous forme hachée.
3. **Connexion persistante :**
 Si `remember=True`, la session reste active après la fermeture du navigateur.
4. **Déconnexion propre :** `logout_user()`
 supprime l'ID utilisateur de la session pour éviter de rester connecté.
5. **Récupération de l'utilisateur connecté :** `get_current_user()`
 permet d'obtenir l'utilisateur actuellement en session pour accéder à ses infos.

3.3 Routes clés

routes/task_routes.py

Dashboard Route

```
task_bp = Blueprint('task', __name__)

@task_bp.route('/dashboard')
def dashboard():
    user = AuthController.get_current_user()
    if not user:
        flash('Please login to access this page', 'danger')
        return redirect(url_for('auth.login'))

    # Get sorting and filtering parameters
    sort_by = request.args.get('sort_by', 'due_date_asc')
    filter_status = request.args.get('filter_status')

    # Get tasks and stats from TaskController
    tasks = TaskController.get_user_tasks(user.id, sort_by, filter_status)
    stats = TaskController.get_dashboard_stats(user.id)

    # Create form instance
    form = TaskForm()

    # Check for modal parameter
    open_modal = request.args.get('openModal')

    return render_template('dashboard.html',
                           tasks=tasks,
                           stats=stats,
                           form=form,
                           current_sort=sort_by,
                           current_filter=filter_status,
                           open_modal=open_modal)
```

1 Checks if the user is logged in

- Calls `AuthController.get_current_user()`.
- If no user is found, redirects to the login page.

2 Retrieves sorting & filtering options

- `sort_by` → Defaults to sorting by `due_date_asc`.
- `filter_status` → Filters tasks based on their status.

3 Fetches tasks & stats

- Calls `TaskController.get_user_tasks(user.id, sort_by, filter_status)`.
- Calls `TaskController.get_dashboard_stats(user.id)`.

4 Creates a new form instance

- `form = TaskForm()` initializes the task creation form.

4 Renders the dashboard template

- Passes tasks, statistics, filters, and form to `dashboard.html`.

Edit Task Route

```
@task_bp.route('/task/edit/<int:task_id>', methods=['POST'])
def edit_task(task_id):
    user = AuthController.get_current_user()
    if not user:
        flash('Please login to access this page', 'danger')
        return redirect(url_for('auth.login'))

    try:
        # Validate CSRF token first
        validate_csrf(request.form.get('csrf_token'))

        success, message = TaskController.update_task(
            task_id,
            user.id,
            title=request.form['title'],
            description=request.form.get('description', ''),
            due_date=request.form.get('due_date'),
            priority=request.form.get('priority', 'medium'),
            status=request.form.get('status', 'pending')
        )

        flash(message, 'success' if success else 'danger')
        return redirect(url_for('task.dashboard'))

    except ValidationError:
        flash('CSRF token missing or invalid', 'danger')
        return redirect(url_for('task.dashboard'))
    except Exception as e:
        flash(f'Error updating task: {str(e)}', 'danger')
        return redirect(url_for('task.dashboard'))
```

- Checks authentication
- Validates CSRF token to prevent attacks
- Calls TaskController.update_task(...) to modify the task
- Handles exceptions and displays flash messages

Add Task Route

```
@task_bp.route('/task/add', methods=['POST'])
def add_task():
    user = AuthController.get_current_user()
    if not user:
        flash('Please login to access this page', 'danger')
        return redirect(url_for('auth.login'))

    form = TaskForm()
    if form.validate_on_submit():
        try:
            success, message = TaskController.create_task(
                user.id,
                form.title.data,
                form.description.data,
                form.due_date.data.strftime('%Y-%m-%d'),
                form.priority.data,
                form.status.data
            )
            flash(message, 'success' if success else 'danger')
        except Exception as e:
            flash(f'Error creating task: {str(e)}', 'danger')
        else:
            for field, errors in form.errors.items():
                for error in errors:
                    flash(f'{field}: {error}', 'danger')

    return redirect(url_for('task.dashboard'))
```

- Checks authentication → Only logged-in users can create tasks.
- Validates the form
→ Uses form.validate_on_submit().
- Calls the controller
→ TaskController.create_task(...) handles business logic.
- Handles errors → Uses try-except to catch errors and display flash messages.
- Redirects to dashboard after adding the task.

Delete Task Route

```
@task_bp.route('/task/delete/<int:task_id>', methods=['POST'])
def delete_task(task_id):
    user = AuthController.get_current_user()
    if not user:
        flash('Please login to access this page', 'danger')
        return redirect(url_for('auth.login'))

    task = Task.query.filter_by(id=task_id, user_id=user.id).first()
    if not task:
        flash('Task not found', 'danger')
        return redirect(url_for('task.dashboard'))

    try:
        db.session.delete(task)
        db.session.commit()
        flash('Task deleted successfully', 'success')
    except Exception as e:
        flash(f'Error deleting task: {str(e)}', 'danger')

    return redirect(url_for('task.dashboard'))
```

- Checks authentication
- Verifies task ownership
- Deletes task from database
- Handles errors properly

```
@task_bp.route('/api/tasks', methods=['GET'])
def api_get_tasks():
```

- Returns a JSON list of all tasks for the logged-in user.

```
@task_bp.route('/api/tasks', methods=['POST'])
def api_create_task():
```

- Creates a new task from JSON data.

```
@task_bp.route('/api/tasks/<int:task_id>', methods=['DELETE'])
def api_delete_task(task_id):
```

- Deletes a task via API request.

3.4 Mécanismes de sécurité

1. Authentification Forte

- **Sessions Serveur** : Stockage des sessions côté serveur (non dans les cookies) avec expiration configurable

- **Jetons CSRF** : Protection intégrée contre les attaques Cross-Site Request Forgery
- **Hachage Sécurisé** : Utilisation d'algorithmes robustes (PBKDF2, bcrypt) pour les mots de passe
- **"Remember Me" Sécurisé** : Jetons uniques à usage unique (pas de stockage direct du mot de passe)

2. Protection des Données

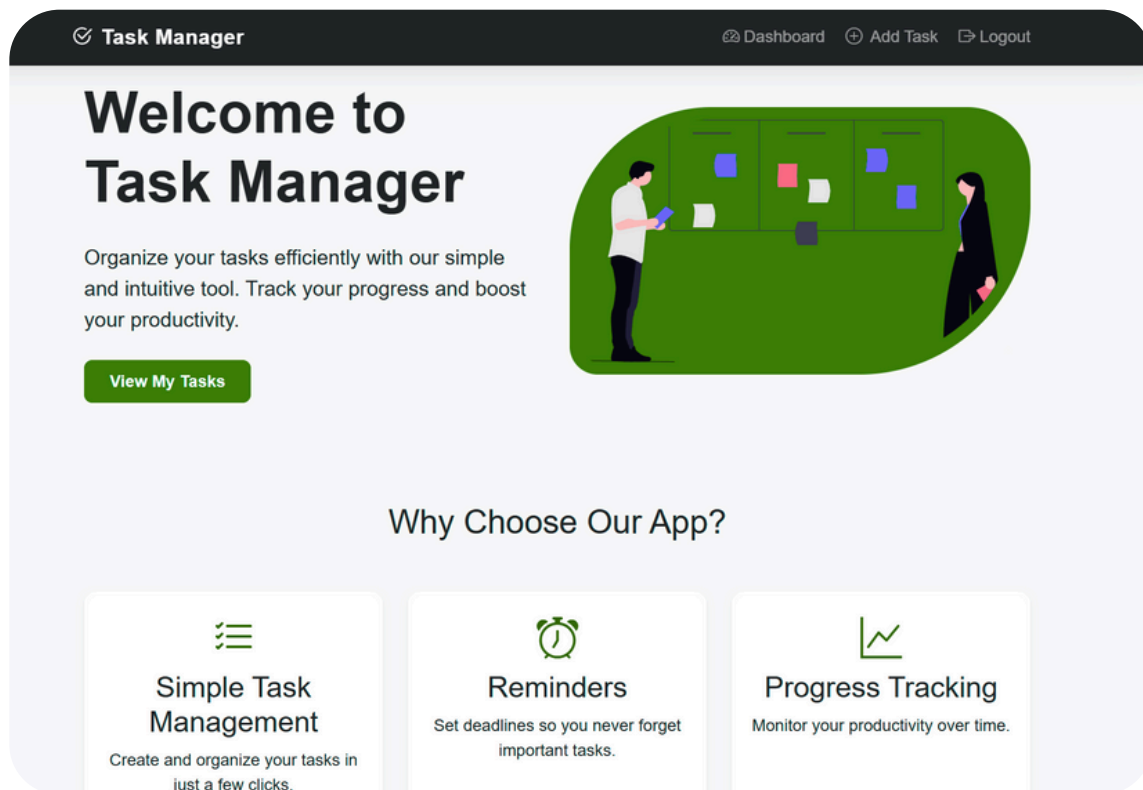
- **Validation Multi-Niveaux** :
 - Côté client (HTML5/Bootstrap)
 - Côté serveur (WTForms + règles métier)
- **Nettoyage des Entrées** : Échappement automatique des données dans les templates Jinja2
- **Protection Contre l'Injection SQL** : Utilisation exclusive de l'ORM SQLAlchemy

3. Sécurité des Communications

- **Cookies Sécurisés** :
 - Flags HttpOnly, Secure, et SameSite
 - Préfixes __Host- et __Secure-
- **Headers HTTP** :
 - HSTS (HTTP Strict Transport Security)
 - CSP (Content Security Policy)
 - Protection contre le sniffing MIME

Capture d'écran de l'application

4.1 Page d'Accueil



- En-tête : Logo "Task Manager" + barre de navigation (Connexion/Inscription)
- Section Principale :
 - Titre : "Bienvenue sur Task Manager"
 - Texte : "Organisez vos tâches efficacement avec notre outil intuitif"
 - Bouton : "Voir mes tâches" (affiche "S'inscrire" si déconnecté)
- Fonctionnalités :
 - a. Gestion simplifiée - "Créez des tâches en quelques clics"
 - b. Rappels - "Définissez des échéances"
 - c. Suivi - "Visualisez votre productivité"

4.2 Écran de Connexion

The screenshot shows a web application interface for a 'Task Manager'. At the top, there is a dark header bar with the text 'Task Manager' on the left and 'Login' and 'Register' links on the right. The main content area is light gray and contains a white login form with a green header labeled 'Login'. The form has two input fields: 'Username' with the value 'admin' and 'Password' with masked characters '.....'. Below the password field is a checkbox labeled 'Remember Me'. A green 'Login' button is positioned below the checkbox. At the bottom of the form, there is a link that says 'Don't have an account? [Register here](#)'.

- Formulaire :
 - Champ "Nom d'utilisateur" (exemple : "admin")
 - Champ "Mot de passe" (masqué)
 - Case "Se souvenir de moi"
 - Bouton "Se connecter" (bleu)
- Lien : "Pas de compte ? S'inscrire ici"

4.3 Écran d'Inscription

Task Manager

Login Register

Register

Username

admin

Email

Password

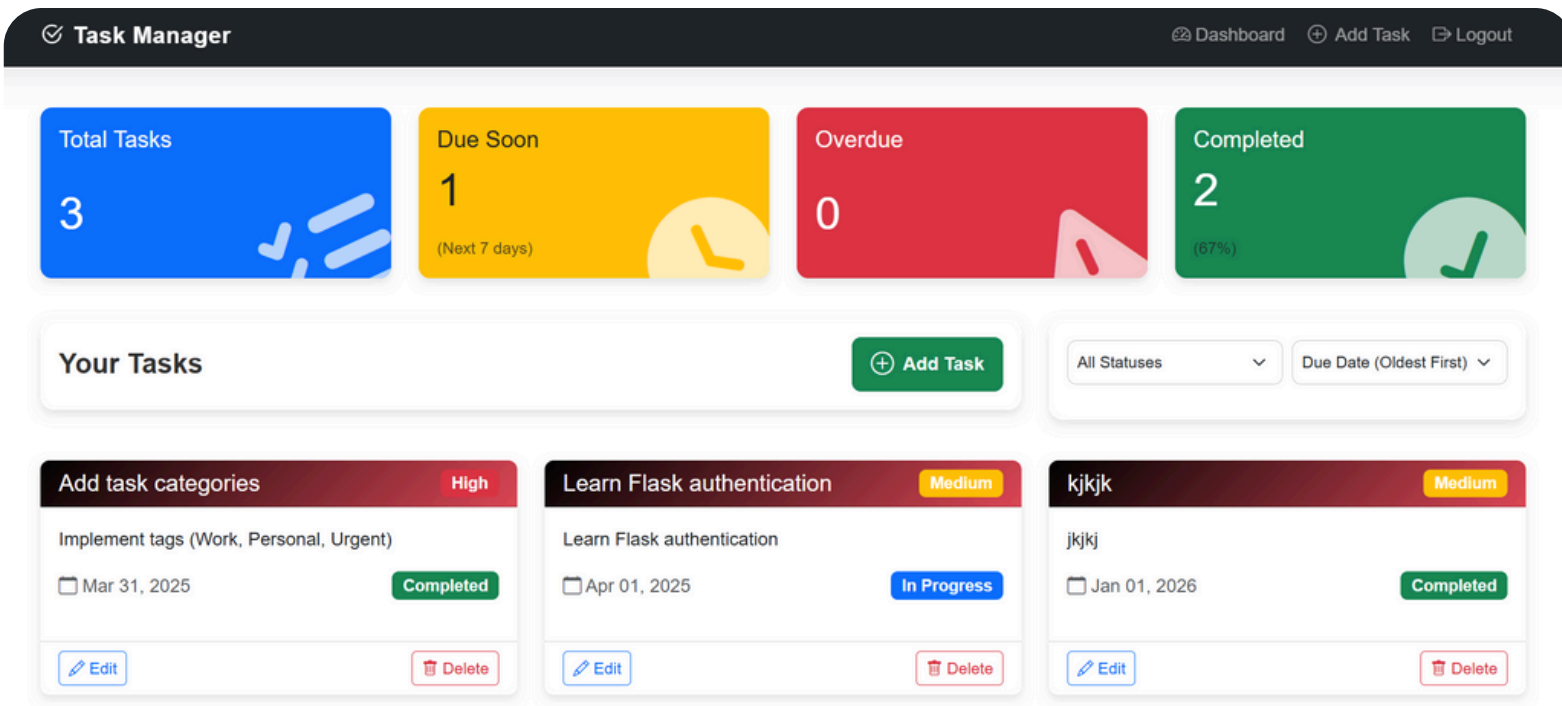
Confirm Password

Register

Already have an account? [Login here](#)

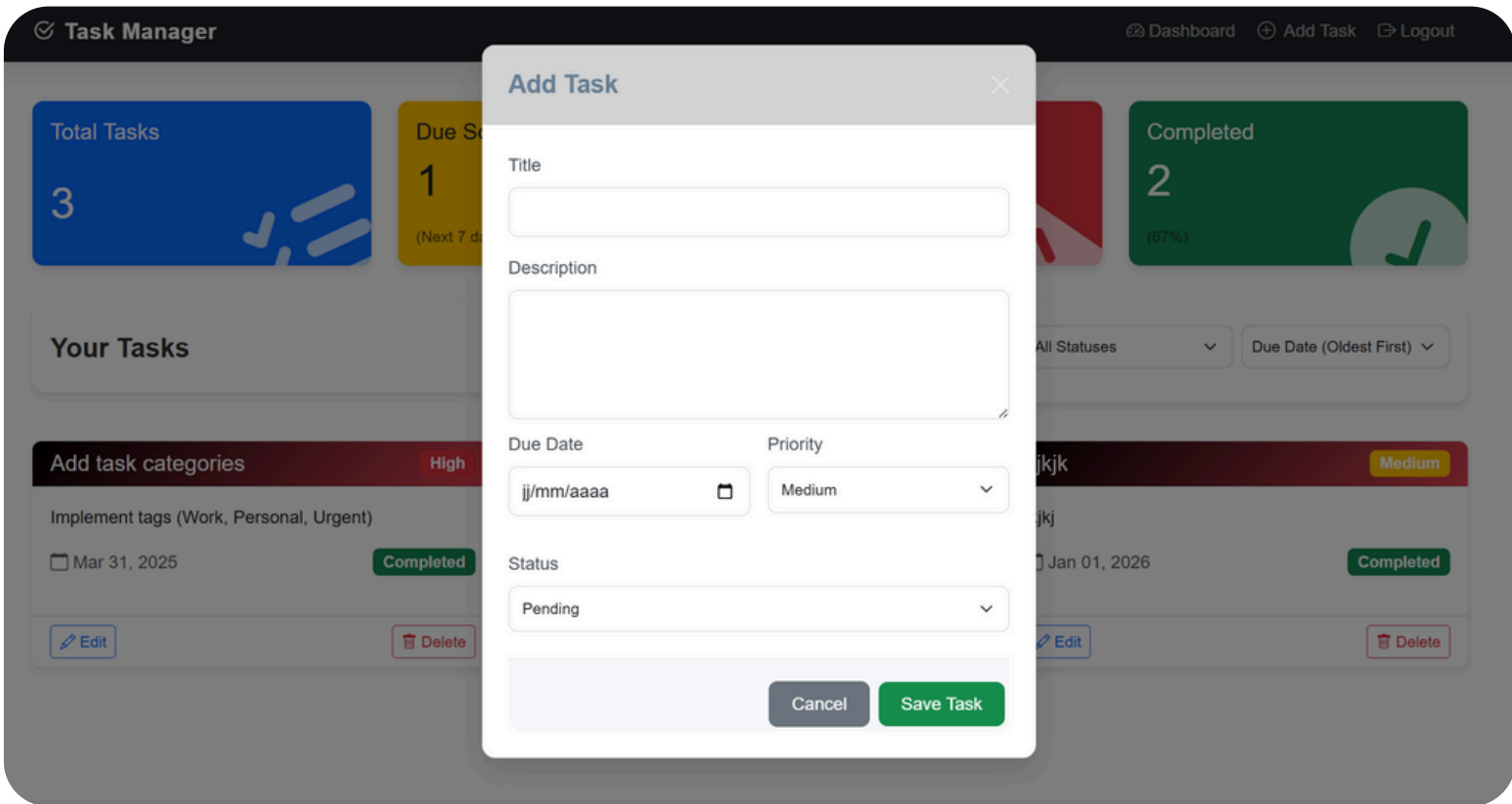
- Champs :
 - Nom d'utilisateur (validation : 4 caractères min)
 - Email (validation format)
 - Mot de passe + Confirmation (indicateur de force)
- Bouton : "S'inscrire" (vert)
- Lien : "Déjà un compte ? Se connecter"

4.4 Tableau de Bord



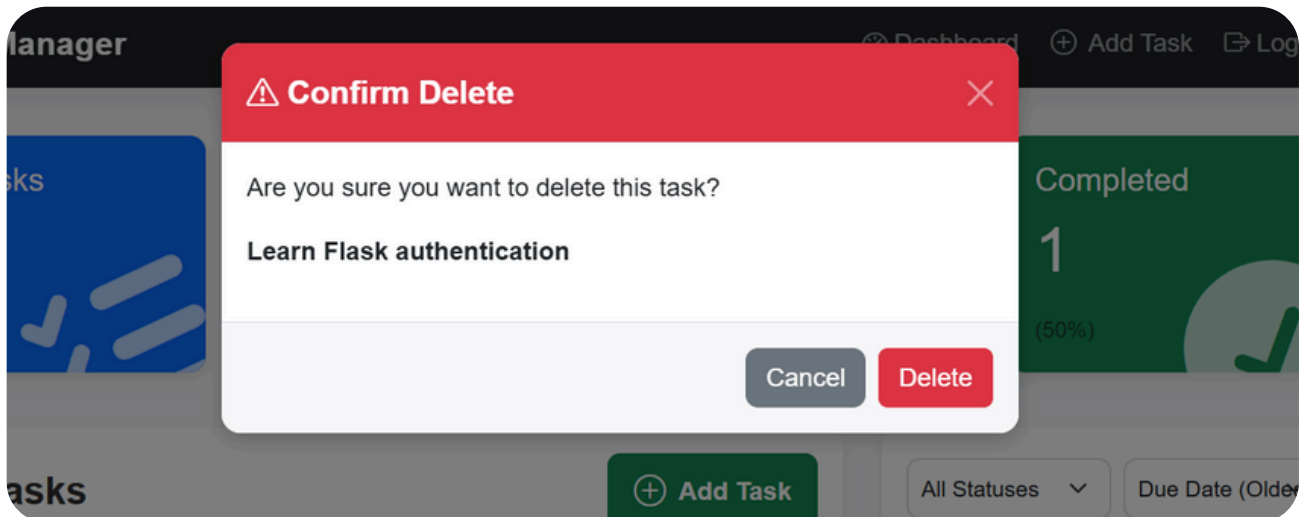
- Statistiques :
 - Tâches totales : 3
 - "Échéance proche" : 1 (badge orange)
 - "En retard" : 0
 - "Terminées" : 2 (67%, barre de progression)
- Liste des tâches avec priorités et dates
- Filtres : "Tous statuts" + "Tri par date"

4.5 Création de Tâche



- Modal avec :
 - Titre (obligatoire)
 - Description
 - Date d'échéance (sélecteur)
 - Priorité (menu déroulant)
 - Boutons "Annuler"/"Enregistrer"

4.6 Confirmation de Suppression



- Message : "Confirmer la suppression ?"
- Aperçu de la tâche concernée
- Boutons "Annuler"/"Supprimer" (rouge)

4.7 Page d'Erreur 404



Oops! Page not found.

The page you're looking for doesn't exist or has been moved.

[Go to Dashboard](#)

- Message : "opps"
- Boutons "Go To Dashboard"



Conclusion

This project demonstrates comprehensive understanding of:

- Python web development
- Database design
- Security principles
- Software architecture patterns