Voici ci-dessous un exemple détaillé de cahier des charges fonctionnel pour une application de reporting de bugs développée avec Django. Ce document vise à décrire l’ensemble des besoins, fonctionnalités et contraintes du projet, afin de servir de référentiel pour l’équipe de développement et les parties prenantes.

**Cahier des charges fonctionnel**

**Application de Reporting de Bugs – Projet Django**

**1. Introduction**

**Contexte et objectif :**  
Dans le cadre de l’amélioration continue de la qualité des applications et de la réactivité face aux problèmes remontés, ce projet a pour but de développer une application web, basée sur le framework Django, permettant aux utilisateurs de signaler, suivre et gérer des bugs ou anomalies.  
**But :**

* Offrir une interface intuitive pour permettre aux utilisateurs (clients, testeurs, collaborateurs) de signaler des incidents de manière claire et structurée.
* Mettre en place un système de suivi et de gestion des bugs accessible aux équipes techniques pour assurer une résolution rapide des problèmes.

**2. Périmètre du projet**

**Fonctionnalités couvertes :**

* Soumission de rapports de bugs (formulaire, champs spécifiques, attachements, etc.)
* Gestion des utilisateurs avec rôles différenciés (utilisateur, testeur, développeur, administrateur)
* Workflow de traitement des bugs (nouvelles déclarations, en cours, résolus, rejetés)
* Notifications (email, éventuellement SMS) liées à l’évolution des tickets
* Dashboard et statistiques (nombre de bugs ouverts, délais de résolution, etc.)
* Recherche et filtrage des bugs
* Historique et traçabilité des actions (journal des modifications)

**Fonctionnalités exclues (pour la première version) :**

* Intégration d’une messagerie instantanée directement dans l’application
* Gestion de versionning des rapports au-delà du suivi de l’historique
* Intégration complexe avec d’autres outils de gestion (ticketing externe) sans API dédiée

**3. Parties prenantes et acteurs**

**Intervenants principaux :**

* **Utilisateur final (Reporter) :** Toute personne pouvant signaler un bug via l’application.
* **Testeur / QA :** Personne en charge de vérifier et reproduire le bug signalé.
* **Développeur :** Responsable de la correction et de la vérification des anomalies.
* **Administrateur :** Supervise le bon fonctionnement de l’application, gère les utilisateurs et la configuration générale.

**Rôles et responsabilités :**

* **Reporter :** Remplit le formulaire de bug, fournit les informations nécessaires (environnement, pas à pas, captures d’écran)
* **Testeur / QA :** Priorise, valide le bug, propose des tests de régression
* **Développeur :** Analyse le rapport, corrige, commente et change le statut du ticket
* **Administrateur :** Gère les droits d’accès, surveille le flux de tickets, contrôle la qualité du suivi

**4. Description des fonctionnalités**

**A. Interface de déclaration d’un bug**

* **Formulaire de saisie :**
  + Champs obligatoires :
    - Titre du bug
    - Description détaillée
    - Étapes pour reproduire le bug
    - Composant / module concerné
    - Environnement (navigateur, version, système d’exploitation)
  + Champs optionnels :
    - Fichiers attachés (captures d’écran, logs)
    - Niveau de gravité (mineur, majeur, critique)
    - Tags/catégories
* **Validation côté client/serveur :**
  + Vérification des champs obligatoires
  + Gestion des formats de fichiers pour les attachements

**B. Gestion et traitement des rapports**

* **Tableau de bord pour les gestionnaires/developpeurs :**
  + Liste et filtres (statut, priorité, date de soumission)
  + Actions :
    - Attribution à un développeur ou à une équipe
    - Changement de statut (nouveau, en cours, en attente, résolu, rejeté)
    - Ajout de commentaires internes
    - Marquage par priorité
* **Suivi de l’historique :**
  + Journal d’activité retraçant :
    - Changements de statut
    - Ajout de commentaires/modifications apportées
    - Chronologie des actions

**C. Gestion des utilisateurs et des rôles**

* **Authentification et autorisation :**
  + Inscription et connexion via formulaire sécurisé
  + Différents niveaux d’accès (gestionnaire, développeur, utilisateur final)
* **Interface de gestion des comptes :**
  + Modification des informations personnelles
  + Gestion des droits et permissions administratives

**D. Notifications et alertes**

* **Système de notifications par email :**
  + Confirmation de la réception du rapport
  + Notifications lors des mises à jour (changement de statut, commentaires)
* **Alertes en temps réel (si implémentable) :**
  + Via websockets pour le dashboard ou intégration de notifications push
* **Logs des notifications :**
  + Archivage et consultation historique des notifications envoyées

**E. Reporting et statistiques**

* **Dashboard analytics :**
  + Graphiques pour visualiser :
    - Nombre de bugs ouverts et résolus par période
    - Temps moyen de traitement des bugs
    - Répartition par priorité et par module
* **Export de données :**
  + CSV, Excel pour analyses externes

**F. Sécurité et confidentialité**

* **Sécurisation des échanges :**
  + Utilisation de HTTPS
  + Contrôle des accès via des rôles et permissions
* **Gestion des données sensibles :**
  + Stockage sécurisé des informations personnelles et des logs
  + Politique de sauvegarde et de récupération des données

**5. Exigences techniques et contraintes**

**A. Technologies et architecture**

* **Framework :**
  + Django (version recommandée : 3.x ou supérieure)
  + Utilisation du pattern MVC (ou MTV propre à Django)
* **Base de données :**
  + PostgreSQL ou MySQL (selon le contexte et l’évolution)
* **Front-end :**
  + Template Django avec possibilité d’intégration d’un framework JS léger (React, Vue.js, ou jQuery) pour des interactions en temps réel (optionnel)
* **Infrastructure et déploiement :**
  + Serveur WSGI (Gunicorn ou uWSGI)
  + Serveur web frontal (Nginx ou Apache)
  + Environnement virtuel pour Python
  + Gestion de la configuration via des variables d’environnement

**B. Performance et scalabilité**

* **Temps de réponse :**
  + Optimisation des requêtes base de données
  + Mise en cache des pages ou requêtes fréquentes
* **Montée en charge :**
  + Architecture modulaire permettant l’ajout de serveurs complémentaires
  + Prévision d’utilisation d’un CDN pour le contenu statique

**C. Sécurité applicative**

* **Authentification sécurisée :**
  + Utilisation des sessions Django sécurisées et d’une gestion fine des permissions
* **Prévention contre les attaques communes :**
  + Protection contre CSRF, XSS, injections SQL
  + Journaux de sécurité et alertes d’activité suspecte

**6. Interfaces et ergonomie**

**A. Maquettes et wireframes (à valider)**

* **Page d’accueil :**
  + Présentation rapide du service et accès aux différents modules (rapport, suivi, statistiques)
* **Page de création d’un rapport de bug :**
  + Formulaire ergonomique avec indicateurs visuels pour les champs obligatoires
  + Interface d’upload d’images et de documents
* **Dashboard de gestion :**
  + Vue synthétisée des rapports en cours, avec filtres et options de recherche
  + Graphiques interactifs pour le reporting
* **Espace personnel utilisateur :**
  + Historique des rapports soumis, suivi des actions

**B. Expérience utilisateur (UX/UI)**

* **Navigation intuitive :**
  + Menu clair et hiérarchisé
  + Réactivité sur mobile et tablette
* **Accessibilité :**
  + Conformité aux standards WCAG (couleurs contrastées, navigation au clavier)
* **Messages et alertes :**
  + Notifications claires et informatives en cas d’erreur ou de validation d’action

**7. Planning et phases de développement**

**A. Phases du projet**

1. **Analyse des besoins et conception :**
   * Finalisation du cahier des charges fonctionnel et technique
   * Réalisation des maquettes et prototypage UX
2. **Développement et tests :**
   * Mise en place de l’architecture Django
   * Développement des fonctionnalités principales (création, gestion, notifications)
   * Tests unitaires et intégration continue
3. **Recette et mise en production :**
   * Phase de bêta test avec utilisateurs pilotes
   * Ajustements et corrections
   * Déploiement sur environnement de production
4. **Maintenance et évolutions :**
   * Suivi des rapports d’incidents
   * Mise à jour régulière et ajout de nouvelles fonctionnalités (optionnel)

**B. Livrables attendus**

* Cahier des charges finalisé
* Code source commenté et versionné (via Git)
* Documentation utilisateur et technique (installation, utilisation, maintenance)
* Environnement de test et de production mis en place

**8. Suivi et gestion de projet**

* **Méthodologie :**
  + Méthode agile (sprints de 2 à 4 semaines) pour itérer et intégrer les retours des utilisateurs
* **Outils de suivi :**
  + Utilisation de systèmes de gestion de projets (JIRA, Trello, ou autre)
  + Réunions de suivi régulières (daily stand-ups, sprint reviews)
* **Indicateurs de qualité :**
  + Taux de résolution des bugs
  + Temps moyen de correction
  + Satisfaction utilisateur (feedback post-déploiement)

**9. Annexes**

* **Diagrammes UML :**
  + Diagramme de cas d’utilisation (Use Case Diagram)
  + Diagramme de classes pour la gestion des entités principales (Bug, Utilisateur, Commentaire)
* **Maquettes graphiques :**
  + Annexer des captures d’écran ou schémas représentant l’interface proposée
* **Spécifications API (si intégration externe) :**
  + Documentation des endpoints pour les notifications ou intégrations tierces

**Conclusion**

Ce cahier des charges fonctionnel constitue un point de départ solide pour la conception d’une application de reporting de bugs sous Django. Il permet d’aligner les besoins des utilisateurs finaux avec les spécifications techniques nécessaires pour un développement efficace. Chaque section pourra être ajustée lors des phases d’analyse détaillée et de prototypage, afin de répondre précisément aux exigences fonctionnelles et aux contraintes techniques du projet.

N’hésitez pas à revenir sur certains points pour affiner les détails ou ajouter des cas spécifiques en fonction de l’évolution des besoins et des retours des futurs utilisateurs.