Système de suivi de tickets (HelpDesk)

Technologies principales : Symfony (PHP), JavaScript, MySQL, HTML, CSS

Equipe de projet: Jaouad Ouhammou
Rachid Moustarhfiri
Badreddine Moussa

1. Contexte et objectifs

L'objectif du projet est de fournir une application web de gestion et de suivi des demandes (tickets) destinée aux équipes de support technique et aux utilisateurs finaux. Le système permettra la création, la priorisation, le suivi des statuts, l'assignation des techniciens et la visualisation via un tableau de bord centralisé.

Bénéfices attendus :

- Centralisation des demandes et historique complet des interventions.
- Meilleure priorisation et réduction des délais de résolution.
- Visibilité opérationnelle via tableaux de bord et rapports.

2. Périmètre fonctionnel

2.1 Utilisateurs / Acteurs

- Utilisateur final (Client) : crée et suit ses tickets.
- Technicien / Agent support : reçoit des tickets, y répond, met à jour le statut.
- Manager : supervise le flux, consulte les tableaux de bord et les rapports.
- Administrateur système : gère les comptes, rôles, configurations et sauvegardes.

2.2 Cas d'utilisation principaux

- Création d'un ticket (formulaire web, champs requis/optionnels).
- Visualisation et recherche de tickets (filtres par statut, priorité, technicien, date).
- Priorisation (Urgent, Élevé, Normal, Faible)-
- Workflow des statuts (Nouveau → En cours → En attente → Résolu → Clos).
- Assignation automatique / manuelle des techniciens.
- Commentaires et échanges (journal de communication) sur chaque ticket.
- Tableau de bord (nombre de tickets ouverts, temps moyen de résolution, tickets par technicien).
- Notifications (email / in-app) pour événements clés (nouveau ticket, assignation, changement de statut).
- Historique complet (audit trail) des modifications.
- Export / rapports (CSV / PDF) pour analyses.

3. Exigences fonctionnelles détaillées

3.1 Gestion des tickets

- F1 : Un utilisateur authentifié peut créer un ticket avec : titre, description, catégorie, priorité, pièces jointes, contact.
- F2 : Le ticket reçoit un identifiant unique et un statut initial `Nouveau`.
- F3 : Les techniciens peuvent changer le statut et ajouter des commentaires. Toute modification génère une entrée dans l'historique.
- F4 : Possibilité d'assigner un ticket à un technicien ; option d'assignation automatique selon règles (charge, compétences).

- F5 : Les tickets peuvent être fermés seulement après validation (option configurable).

3.2 Priorisation et SLA

- F6 : Définir des niveaux de priorité et des SLA associés (ex. : Urgent = réponse < 1h, Résolution < 4h).

- F7 : Indicateur visuel sur le tableau de bord pour les SLA approchants et violés.

3.3 Recherche & filtrage

- F8 : Recherche textuelle sur titre, description, commentaires.

- F9 : Filtres multi-critères (statut, priorité, technicien, période, catégorie).

3.4 Notifications

- F10 : Notifications par email et in-app pour : création, assignation, commentaire, changement de statut, SLA violé.

- F11 : Configuration des notifications par utilisateur (activer/désactiver types).

3.5 Sécurité & authentification

- F12 : Authentification par mot de passe

- F13 : Gestion des rôles (Admin, Manager, Technicien, Utilisateur) avec permissions granulaires.

3.6 Interface utilisateur

- F15 : Interface responsive (desktop/tablette/mobile).

- F16 : Tableau de bord configurable par rôle.

- F17 : Formulaire de création simple et un mode avancé pour champs supplémentaires.

3.7 Administration

- F18 : Gestion des utilisateurs, des rôles, des catégories, des priorités et des workflows.

- F19 : Sauvegarde/restauration des données (sauvegardes MySQL).

4. Architecture technique proposée

4.1 Backend

- Framework : Symfony (PHP)
- Base de données : MySQL (schéma relationnel)
- Authentification : Symfony Security

4.2 Frontend

- HTML / CSS
- JavaScript
- Utilisation d'APIs REST du backend.

4.3 Infrastructure

- Serveur Apache via XAMPP

5. Modèle de données (schéma simplifié)

Tables principales proposées :

- `user` (id, nom, email, password_hash, role_id, actif, created_at)
- `role` (id, nom, permissions)
- `ticket` (id, reference, titre, description, categorie_id, priorite_id, statut_id, creator_id, assignee_id, sla_due, created_at, updated_at)
- `ticket_comment` (id, ticket_id, author_id, message, created_at)
- `ticket_attachment` (id, ticket_id, filename, path, uploaded_by, uploaded_at)

- `ticket_history` (id, ticket_id, field, old_value, new_value, changed_by, changed_at)

- `category`, `priority`, `status`

6. API (exemples d'endpoints)

- `POST /api/tickets` — Créer un ticket.
- `GET /api/tickets` — Lister / filtrer les tickets.
- `GET /api/tickets/{id}` — Détails d'un ticket.
- `PUT /api/tickets/{id}` — Mettre à jour (statut, assignation, description).
- `POST /api/tickets/{id}/comments` — Ajouter un commentaire.
- `POST /api/tickets/{id}/attachments` — Joindre un fichier.
- `GET /api/dashboard` — KPIs pour le tableau de bord.

7. Tests et recette

- Tests unitaires backend

- Tests fonctionnels

- Plan de recette : jeux de données, scénarios critiques (création, assignation, SLA, fermeture), tests de montée en charge basique.

8. Déploiement et exploitation

- Environnements : `dev`, `staging`, `production`.

- CI/CD : pipeline pour tests et déploiement (GitLab CI / GitHub Actions).

- Monitoring : logs applicatifs, alertes sur erreurs 5xx et violations SLA.

- Procédure de sauvegarde MySQL et rotation des fichiers joints.

9. Critères d'acceptation

- Création d'un ticket et suivi du cycle complet (de `Nouveau` à `Clos`) validé.
- Assignation et notification opérationnelles.
- Tableau de bord affichant KPIs et graphiques avec données réelles.
- Tests unitaires et d'intégration couvrant les fonctionnalités critiques.
- Documentation technique et guide d'utilisation pour les utilisateurs finaux.