

PM

MODULE PROJECT MANAGEMENT

Architecture – Project Management

architecture_jira_project_management.md

HR_ONIAN · Spécification Technique · Février 2026

Architecture JIRA-like pour la Gestion de Projet et Suivi des Tâches

Vue d'ensemble

Ce document présente une architecture complète pour un système de gestion de projet et suivi des tâches inspiré de JIRA, intégré dans l'application Django HR_ONIAN existante. Le système se concentre sur la création de "tickets" pour chaque tâche et nouvelle fonctionnalité, avec un workflow défini et une gestion de backlog.

Modèles de Données

1. Modèle Client (JRCLIENT)

```

class JRClient(models.Model):
    """Modèle pour la gestion des clients"""

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    code_client = models.CharField(
        max_length=20,
        unique=True,
        verbose_name="Code Client",
        editable=False
    )
    raison_sociale = models.CharField(
        max_length=200,
        verbose_name="Raison Sociale"
    )

    # Informations de contact
    contact_principal = models.CharField(
        max_length=100,
        verbose_name="Contact principal"
    )
    email_contact = models.EmailField(
        verbose_name="Email contact"
    )
    telephone_contact = models.CharField(
        max_length=20,
        blank=True,
        null=True,
        verbose_name="Téléphone"
    )

    # Adresse
    adresse = models.TextField(
        blank=True,
        null=True,
        verbose_name="Adresse"
    )
    code_postal = models.CharField(
        max_length=10,
        blank=True,
        null=True,
        verbose_name="Code Postal"
    )
    ville = models.CharField(
        max_length=100,
        blank=True,
        null=True,
        verbose_name="Ville"
    )
    pays = models.CharField(
        max_length=50,
        default="France",
        verbose_name="Pays"
    )

    # Informations de facturation
    numero_tva = models.CharField(
        max_length=20,
        blank=True,
        null=True,
        verbose_name="Numéro TVA"
    )
    conditions_paiement = models.TextField(
        blank=True,
        null=True,
        verbose_name="Conditions de paiement"
    )

    # Statut
    STATUT_CHOICES = [
        ('ACTIF', 'Actif'),
        ('INACTIF', 'Inactif'),
        ('SUSPENDU', 'Suspendsu'),
    ]
    statut = models.CharField(
        max_length=20,
        choices=STATUT_CHOICES,
        default='ACTIF'
    )

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

```

```

class Meta:
    verbose_name = "Client"
    verbose_name_plural = "Clients"
    ordering = ['raison_sociale']

    def __str__(self):
        return f"{self.code_client} - {self.raison_sociale}"

    def save(self, *args, **kwargs):
        # Génération automatique du code client
        if not self.code_client:
            prefix = "CL"
            last_client = JRClient.objects.filter(
                code_client__startswith=prefix
            ).order_by('code_client').last()

            if last_client:
                try:
                    last_num = int(last_client.code_client.split('-')[-1])
                    new_num = last_num + 1
                except (ValueError, IndexError):
                    new_num = 1
            else:
                new_num = 1

            self.code_client = f"{prefix}-{new_num:04d}"

        super().save(*args, **kwargs)

@property
def nombre_projets(self):
    """Retourne le nombre de projets du client"""
    return self.projets.count()

@property
def chiffre_affaires_total(self):
    """Calcule le chiffre d'affaires total avec ce client"""
    return sum(projet.montant_total or 0 for projet in self.projets.all())

```

2. Modèle Projet (JRPROJECT)

```

class JRProject(models.Model):
    """Modèle pour la gestion des projets"""

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    code = models.CharField(max_length=20, unique=True, verbose_name="Code Projet")
    nom = models.CharField(max_length=200, verbose_name="Nom du projet")
    description = models.TextField(blank=True, null=True)

    # Gestion du projet
    client = models.ForeignKey(
        JRClient,
        on_delete=models.PROTECT,
        related_name='projets',
        verbose_name="Client"
    )

    chef_projet = models.ForeignKey(
        'employee.ZY00',
        on_delete=models.SET_NULL,
        null=True,
        related_name='projets_diriges',
        verbose_name="Chef de projet"
    )

    # Informations financières
    montant_total = models.DecimalField(
        max_digits=12,
        decimal_places=2,
        null=True,
        blank=True,
        verbose_name="Montant total (€)"
    )

    # Dates et statut
    date_debut = models.DateField(verbose_name="Date de début")
    date_fin_prevue = models.DateField(verbose_name="Date de fin prévue")
    date_fin_reelle = models.DateField(null=True, blank=True)

    STATUT_CHOICES = [
        ('PLANIFIÉ', 'Planifié'),
        ('ACTIF', 'Actif'),
        ('EN_PAUSE', 'En pause'),
        ('TERMINÉ', 'Terminé'),
        ('ANNULÉ', 'Annulé'),
    ]
    statut = models.CharField(max_length=20, choices=STATUT_CHOICES, default='PLANIFIÉ')

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Projet"
        verbose_name_plural = "Projets"
        ordering = ['-created_at']

    def __str__(self):
        return f"{self.code} - {self.nom}"

    @property
    def progression(self):
        """Calcule la progression du projet basée sur les tickets"""
        total_tickets = self.tickets.count()
        if total_tickets == 0:
            return 0
        tickets_termines = self.tickets.filter(statut='TERMINÉ').count()
        return round((tickets_termines / total_tickets) * 100, 2)

```

2. Modèle Ticket (JRTICKET)

```

class JRTicket(models.Model):
    """Modèle principal pour les tickets/tâches"""

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    code = models.CharField(max_length=20, unique=True, verbose_name="Code Ticket")

    # Liaison avec le projet
    projet = models.ForeignKey(
        JRProject,
        on_delete=models.CASCADE,
        related_name='tickets',
        verbose_name="Projet"
    )

    # Informations de base
    titre = models.CharField(max_length=200, verbose_name="Titre du ticket")
    description = models.TextField(verbose_name="Description détaillée")

    # Priorité
    PRIORITE_CHOICES = [
        ('BASSE', 'Basse'),
        ('MOYENNE', 'Moyenne'),
        ('HAUTE', 'Haute'),
        ('CRITIQUE', 'Critique'),
    ]
    priorite = models.CharField(
        max_length=20,
        choices=PRIORITE_CHOICES,
        default='MOYENNE',
        verbose_name="Priorité"
    )

    # Workflow
    STATUT_CHOICES = [
        ('OUVERT', 'Ouvert'),
        ('EN_COURS', 'En cours'),
        ('EN_REVUE', 'En revue'),
        ('TERMINÉ', 'Terminé'),
    ]
    statut = models.CharField(
        max_length=20,
        choices=STATUT_CHOICES,
        default='OUVERT',
        verbose_name="Statut"
    )

    # Assignation
    assigne = models.ForeignKey(
        'employee.ZY00',
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='tickets_assignes',
        verbose_name="Assigné à"
    )

    # Type de ticket
    TYPE_CHOICES = [
        ('TACHE', 'Tâche'),
        ('AMELIORATION', 'Amélioration'),
        ('NOUVELLE_FONCTIONNALITE', 'Nouvelle fonctionnalité'),
        ('BUG', 'Bug'),
    ]
    type_ticket = models.CharField(
        max_length=30,
        choices=TYPE_CHOICES,
        default='TACHE',
        verbose_name="Type de ticket"
    )

    # Gestion du temps
    estimation_heures = models.DecimalField(
        max_digits=5,
        decimal_places=2,
        null=True,
        blank=True,
        verbose_name="Estimation (heures)"
    )
    temps_passe = models.DecimalField(
        max_digits=5,
        decimal_places=2,
        default=0,
    )

```

```

        verbose_name="Temps passé (heures)"
    )

    # Dates
    date_echeance = models.DateField(
        null=True,
        blank=True,
        verbose_name="Date d'échéance"
    )
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # Backlog
    dans_backlog = models.BooleanField(
        default=False,
        verbose_name="Dans le backlog"
    )
    ordre_backlog = models.PositiveIntegerField(
        default=0,
        verbose_name="Ordre dans le backlog"
    )

    class Meta:
        verbose_name = "Ticket"
        verbose_name_plural = "Tickets"
        ordering = ['-created_at']

    def __str__(self):
        return f"{self.code} - {self.titre}"

    def save(self, *args, **kwargs):
        # Génération automatique du code si non fourni
        if not self.code:
            prefix = "TK"
            if self.projet:
                prefix = self.projet.code
            last_ticket = JRTicket.objects.filter(
                code__startswith=prefix
            ).order_by('code').last()

            if last_ticket:
                try:
                    last_num = int(last_ticket.code.split('-')[-1])
                    new_num = last_num + 1
                except (ValueError, IndexError):
                    new_num = 1
            else:
                new_num = 1

            self.code = f"{prefix}-{new_num:04d}"
        super().save(*args, **kwargs)

```

3. Modèle Commentaire (JRCOMMENTAIRE)

```

class JRCommentaire(models.Model):
    """Modèle pour les commentaires sur les tickets"""

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    ticket = models.ForeignKey(
        JRTicket,
        on_delete=models.CASCADE,
        related_name='commentaires',
        verbose_name="Ticket"
    )

    auteur = models.ForeignKey(
        'employee.ZY00',
        on_delete=models.CASCADE,
        related_name='commentaires_tickets',
        verbose_name="Auteur"
    )

    contenu = models.TextField(verbose_name="Contenu du commentaire")

    # Mentions d'autres utilisateurs
    mentions = models.ManyToManyField(
        'employee.ZY00',
        blank=True,
        related_name='mentions_commentaires',
        verbose_name="Mentions"
    )

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Commentaire"
        verbose_name_plural = "Commentaires"
        ordering = ['created_at']

    def __str__(self):
        return f"Commentaire de {self.auteur} sur {self.ticket.code}"

```

4. Modèle Pièce Jointe (JRPIECEJOINTE)

```

class JRPieceJointe(models.Model):
    """Modèle pour les pièces jointes des tickets"""

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    ticket = models.ForeignKey(
        JRTicket,
        on_delete=models.CASCADE,
        related_name='pieces_jointes',
        verbose_name="Ticket"
    )

    fichier = models.FileField(
        upload_to='tickets_pieces_jointes/%Y/%m/',
        validators=[
            FileExtensionValidator(
                allowed_extensions=[
                    'pdf', 'doc', 'docx', 'xls', 'xlsx',
                    'jpg', 'jpeg', 'png', 'gif', 'zip', 'rar'
                ]
            )
        ],
        verbose_name="Fichier"
    )

    nom_original = models.CharField(max_length=255, verbose_name="Nom original")
    taille = models.PositiveIntegerField(verbose_name="Taille (octets)")
    type_mime = models.CharField(max_length=100, verbose_name="Type MIME")

    uploaded_by = models.ForeignKey(
        'employee.ZY00',
        on_delete=models.SET_NULL,
        null=True,
        verbose_name="Uploadé par"
    )

    uploaded_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = "Pièce jointe"
        verbose_name_plural = "Pièces jointes"
        ordering = ['-uploaded_at']

    def __str__(self):
        return f"{self.nom_original} - {self.ticket.code}"

    @property
    def taille_formattee(self):
        """Retourne la taille formatée en KB/MB"""
        taille = self.taille
        for unit in ['o', 'Ko', 'Mo', 'Go']:
            if taille < 1024.0:
                return f"{taille:.1f} {unit}"
            taille /= 1024.0
        return f"{taille:.1f} To"

```

5. Modèle Historique des Changements (JRHISTORIQUE)

```

class JRHistorique(models.Model):
    """Modèle pour suivre l'historique des changements sur les tickets"""

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    ticket = models.ForeignKey(
        JRTicket,
        on_delete=models.CASCADE,
        related_name='historique',
        verbose_name="Ticket"
    )

    utilisateur = models.ForeignKey(
        'employee.ZY00',
        on_delete=models.CASCADE,
        verbose_name="Utilisateur"
    )

    TYPE_CHANGEMENT_CHOICES = [
        ('CREATION', 'Création'),
        ('MODIFICATION', 'Modification'),
        ('STATUT', 'Changement de statut'),
        ('ASSIGNATION', 'Réassignation'),
        ('PRIORITE', 'Changement de priorité'),
        ('COMMENTAIRE', 'Ajout de commentaire'),
        ('PIECE_JOINTE', 'Ajout de pièce jointe'),
    ]

    type_changement = models.CharField(
        max_length=30,
        choices=TYPE_CHANGEMENT_CHOICES,
        verbose_name="Type de changement"
    )

    champ_modifie = models.CharField(
        max_length=50,
        null=True,
        blank=True,
        verbose_name="Champ modifié"
    )

    ancienne_valeur = models.TextField(
        null=True,
        blank=True,
        verbose_name="Ancienne valeur"
    )

    nouvelle_valeur = models.TextField(
        null=True,
        blank=True,
        verbose_name="Nouvelle valeur"
    )

    description = models.TextField(verbose_name="Description du changement")

    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = "Historique"
        verbose_name_plural = "Historiques"
        ordering = ['-created_at']

    def __str__(self):
        return f"{self.ticket.code} - {self.type_changement} par {self.utilisateur}"

```

```

### 7. Modèle Imputation de Temps (JRIMPUTATION)

```python
class JRImputation(models.Model):
 """Modèle pour les imputations de temps sur les tickets"""

 id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)

 # Liaison avec l'employé et le ticket
 employe = models.ForeignKey(
 'employee.ZY00',
 on_delete=models.CASCADE,
 related_name='imputations',
 verbose_name="Employé"
)

 ticket = models.ForeignKey(
 JRTicket,
 on_delete=models.CASCADE,
 related_name='imputations',
 verbose_name="Ticket"
)

 # Période d'imputation
 date_imputation = models.DateField(
 verbose_name="Date d'imputation"
)

 # Temps passé
 heures = models.DecimalField(
 max_digits=5,
 decimal_places=2,
 verbose_name="Heures travaillées"
)

 minutes = models.PositiveIntegerField(
 default=0,
 verbose_name="Minutes supplémentaires"
)

 # Description du travail effectué
 description = models.TextField(
 verbose_name="Description du travail"
)

 # Type d'activité
 TYPE_ACTIVITE_CHOICES = [
 ('DEVELOPPEMENT', 'Développement'),
 ('ANALYSE', 'Analyse'),
 ('TEST', 'Test'),
 ('REUNION', 'Réunion'),
 ('DOCUMENTATION', 'Documentation'),
 ('SUPPORT', 'Support'),
 ('AUTRE', 'Autre'),
]

 type_activite = models.CharField(
 max_length=30,
 choices=TYPE_ACTIVITE_CHOICES,
 default='DEVELOPPEMENT',
 verbose_name="Type d'activité"
)

 # Validation
 STATUT_VALIDATION_CHOICES = [
 ('EN_ATTENTE', 'En attente de validation'),
 ('VALIDE', 'Validé'),
 ('REJETE', 'Rejeté'),
]

 statut_validation = models.CharField(
 max_length=20,
 choices=STATUT_VALIDATION_CHOICES,
 default='EN_ATTENTE',
 verbose_name="Statut de validation"
)

 valide_par = models.ForeignKey(
 'employee.ZY00',
 on_delete=models.SET_NULL,
 null=True,
```

```

```

        blank=True,
        related_name='imputations_validees',
        verbose_name="Validé par"
    )

date_validation = models.DateTimeField(
    null=True,
    blank=True,
    verbose_name="Date de validation"
)

commentaire_validation = models.TextField(
    blank=True,
    null=True,
    verbose_name="Commentaire de validation"
)

created_at = models.DateTimeField(auto_now_add=True)
updated_at = models.DateTimeField(auto_now=True)

class Meta:
    verbose_name = "Imputation de temps"
    verbose_name_plural = "Imputations de temps"
    ordering = ['-date_imputation', '-created_at']
    unique_together = ['employe', 'ticket', 'date_imputation']

def __str__(self):
    return f"{self.employe} - {self.ticket.code} - {self.date_imputation} - {self.total_heures}h"

@property
def total_heures(self):
    """Calcule le total des heures (heures + minutes)"""
    return float(self.heures) + (self.minutes / 60)

@property
def total_heures_formatte(self):
    """Retourne le total formaté HH:MM"""
    total_minutes = int(self.total_heures * 60)
    heures = total_minutes // 60
    minutes = total_minutes % 60
    return f"{heures:02d}:{minutes:02d}"

def valider(self, valide_par, commentaire=""):
    """Valide l'imputation"""
    self.statut_validation = 'VALIDE'
    self.valide_par = valide_par
    self.date_validation = timezone.now()
    self.commentaire_validation = commentaire
    self.save()

    # Mettre à jour le temps passé sur le ticket
    self.mettre_a_jour_temps_ticket()

def rejeter(self, valide_par, commentaire):
    """Rejette l'imputation"""
    self.statut_validation = 'REJETE'
    self.valide_par = valide_par
    self.date_validation = timezone.now()
    self.commentaire_validation = commentaire
    self.save()

def mettre_a_jour_temps_ticket(self):
    """Met à jour le temps total passé sur le ticket"""
    if self.statut_validation == 'VALIDE':
        total_temps = JRImputation.objects.filter(
            ticket=self.ticket,
            statut_validation='VALIDE'
        ).aggregate(
            total=models.Sum(
                models.F('heures') + models.F('minutes') / 60.0
            )
        )['total'] or 0

        self.ticket.temps_passe = total_temps
        self.ticket.save()

```

8. Modèle Sprint (JRSpint)

```

class JRSpint(models.Model):
    """Modèle pour la gestion des sprints"""

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    nom = models.CharField(max_length=100, verbose_name="Nom du sprint")
    description = models.TextField(blank=True, null=True)

    projet = models.ForeignKey(
        JRProject,
        on_delete=models.CASCADE,
        related_name='sprints',
        verbose_name="Projet"
    )

    date_debut = models.DateField(verbose_name="Date de début")
    date_fin = models.DateField(verbose_name="Date de fin")

    STATUT_CHOICES = [
        ('PLANIFIE', 'Planifié'),
        ('ACTIF', 'Actif'),
        ('TERMINÉ', 'Terminé'),
    ]
    statut = models.CharField(
        max_length=20,
        choices=STATUT_CHOICES,
        default='PLANIFIE'
    )

    tickets = models.ManyToManyField(
        JRTicket,
        blank=True,
        related_name='sprints',
        verbose_name="Tickets du sprint"
    )

    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Sprint"
        verbose_name_plural = "Sprints"
        ordering = ['-date_debut']

    def __str__(self):
        return f"{self.nom} - {self.projet.code}"

    @property
    def duree_jours(self):
        """Calcule la durée du sprint en jours"""
        return (self.date_fin - self.date_debut).days + 1

    @property
    def progression(self):
        """Calcule la progression du sprint"""
        total_tickets = self.tickets.count()
        if total_tickets == 0:
            return 0
        tickets_termines = self.tickets.filter(statut='TERMINÉ').count()
        return round((tickets_termines / total_tickets) * 100, 2)

```

Workflow Défini

Cycle de Vie d'un Ticket

1. **OUVERT**
2. Le ticket est créé et initial
3. Peut être assigné ou non
4. Peut être déplacé vers le backlog

5. **EN_COURS**

6. Le ticket est en cours de développement
7. Doit être assigné à une personne

8. Le temps passé commence à être tracked

9. EN_REVUE

10. Le développement est terminé

11. En attente de validation/review

12. Peut retourner en EN_COURS si des modifications sont nécessaires

13. TERMINE

14. Le ticket est validé et terminé

15. Plus aucune modification possible

16. Le temps total est finalisé

Règles de Transition

- **OUVERT → EN_COURS:** Nécessite une assignation obligatoire
- **EN_COURS → EN_REVUE:** Le temps passé doit être documenté
- **EN_REVUE → TERMINE:** Validation requise (chef de projet ou assigné)
- **EN_REVUE → EN_COURS:** Possible si des modifications sont requises
- **OUVERT → TERMINE:** Possible pour les tickets très simples

Gestion du Backlog

Fonctionnalités du Backlog

1. Priorisation des Tickets

2. Ordre personnalisable via drag & drop

3. Filtres par priorité, type, assigné

4. Recherche plein texte

5. Planning des Sprints

6. Sélection des tickets du backlog

7. Estimation de la capacité

8. Création automatique des sprints

9. Vue Kanban

10. Colonnes: Backlog, À faire, En cours, En revue, Terminé

11. Drag & drop entre colonnes

12. Limites WIP (Work In Progress)

Architecture Technique

Structure des Applications Django

```

project_management/
├── __init__.py
├── admin.py
├── apps.py
├── models.py      # Tous les modèles définis ci-dessus
└── views/
    ├── __init__.py
    ├── client_views.py      # Vues pour la gestion des clients
    ├── projet_views.py      # Vues pour la gestion des projets
    ├── ticket_views.py      # Vues pour la gestion des tickets
    ├── imputation_views.py  # Vues pour les imputations de temps
    ├── backlog_views.py     # Vues pour la gestion du backlog
    ├── sprint_views.py      # Vues pour la gestion des sprints
    └── api_views.py         # API endpoints
├── forms/
    ├── __init__.py
    ├── client_forms.py      # Formulaires pour les clients
    ├── projet_forms.py      # Formulaires pour les projets
    ├── ticket_forms.py      # Formulaires pour les tickets
    ├── imputation_forms.py  # Formulaires pour les imputations
    └── sprint_forms.py      # Formulaires pour les sprints
└── services/
    ├── __init__.py
    ├── client_service.py    # Logique métier des clients
    ├── ticket_service.py    # Logique métier des tickets
    ├── imputation_service.py # Logique des imputations
    ├── workflow_service.py   # Gestion du workflow
    └── notification_service.py # Notifications
└── templates/
    ├── client/
    │   ├── client_list.html
    │   ├── client_detail.html
    │   └── client_form.html
    ├── projet/
    │   ├── projet_list.html
    │   ├── projet_detail.html
    │   └── projet_form.html
    ├── ticket/
    │   ├── ticket_list.html
    │   ├── ticket_detail.html
    │   ├── ticket_form.html
    │   └── ticket_kanban.html
    ├── imputation/
    │   ├── imputation_list.html
    │   ├── imputation_form.html
    │   ├── validation_list.html
    │   └── rapports_temps.html
    ├── backlog/
    │   └── backlog.html
    └── sprint/
        ├── sprint_list.html
        ├── sprint_detail.html
        └── sprint_board.html
└── static/
    └── css/
        └── project_management.css
            └── kanban.css
└── urls.py

```

Intégration avec les Modules Existantes

1. Module Employee

- Utilisation du modèle `ZY00` pour les utilisateurs
- Intégration avec les permissions existantes
- Gestion des rôles (Chef de projet, Développeur, etc.)

```

### 2. Module Client (intégré)
- Gestion complète des clients avec informations de contact et facturation
- Suivi des projets par client
- Historique et chiffre d'affaires par client

### 3. Module Imputation de Temps (intégré)
- Saisie du temps passé sur les tickets avec validation
- Types d'activité prédefinis (développement, analyse, test, etc.)
- Workflow de validation (en attente → validé/rejeté)
- Rapports de temps par projet/client/employé
- Export des données temporelles

### 4. Module Notifications
- Utilisation du système de notification existant
- Alertes sur les changements de statut
- Notifications pour les assignations
- Rappels d'imputation de temps

## Fonctionnalités Clés

### 1. Tableau de Bord Principal
- Vue d'ensemble des projets actifs
- Statistiques des tickets par statut
- Graphiques de progression
- Tickets récents et urgents

### 2. Gestion des Tickets
- Création et modification de tickets
- Interface drag & drop pour le changement de statut
- Historique complet des modifications
- Système de commentaires avec mentions

### 4. Gestion des Imputations de Temps
- Saisie rapide du temps avec formulaire optimisé
- Validation des imputations par les chefs de projet
- Vue calendrier des imputations
- Rapports détaillés par employé/projet/période
- Export Excel/PDF des temps

### 5. Backlog et Planning
- Vue backlog avec priorisation
- Interface de planning des sprints
- Estimation de la charge de travail
- Vue Kanban interactive

### 6. Rapports et Analytics
- Rapports de progression par projet
- Analyse des temps passés
- Productivité par employé
- Tendances et KPIs
- Rapports financiers par client/projet

### 7. API REST
- Endpoints pour l'intégration externe
- Support pour les applications mobiles
- Webhooks pour les notifications

## Sécurité et Permissions

### Rôles et Permissions

1. **Administrateur**
- Accès complet à toutes les fonctionnalités
- Gestion des utilisateurs et permissions

2. **Chef de Projet**
- Création et gestion des projets
- Assignation des tickets
- Validation des tickets terminés

3. **Développeur**
- Création de tickets
- Mise à jour des statuts
- Ajout de commentaires et pièces jointes

4. **Observateur**
- Lecture seule sur les projets assignés
- Accès aux rapports

### Contrôles d'Accès
- Vérification des permissions au niveau des vues

```

```

- Filtrage des données selon les rôles
- Audit trail complet des actions

## Déploiement et Configuration

### Configuration Requise
- Django 4.2+
- PostgreSQL 13+
- Redis pour le cache et les sessions
- Celery pour les tâches asynchrones

### Variables d'Environnement
```python
Configuration du module de gestion de projet
PROJECT_MANAGEMENT_ENABLED = True
PROJECT_MANAGEMENT_DEFAULT_PRIORITE = 'MOYENNE'
PROJECT_MANAGEMENT_MAX_FILE_SIZE = 10 * 1024 * 1024 # 10MB
PROJECT_MANAGEMENT_ALLOWED_EXTENSIONS = ['pdf', 'doc', 'docx', 'xls', 'xlsx', 'jpg', 'jpeg', 'png']
```

```

Conclusion

Cette architecture fournit une base solide pour implémenter un système de gestion de projet et suivi des tâches complet dans l'application HR_ONIAN. Elle s'intègre harmonieusement avec les modules existants tout en offrant des fonctionnalités robustes inspirées de JIRA.

La modularité de l'architecture permet une évolution progressive et une maintenance aisée. Le système est conçu pour être évolutif et adaptable aux besoins spécifiques de l'organisation.

Tests minimaux (recommandé pour commencer)

```
python manage.py test project_management.tests.test_minimal --verbosity=2
```

Tests des modèles

```
python manage.py test project_management.tests.test_models --verbosity=2
```

Tous les tests

```
python manage.py test project_management --verbosity=2
```