Plan d'Implémentation - Système de Gestion des Notes

1. Vue d'ensemble

Ce document détaille le plan **technique** pour implémenter les fonctionnalités du cahier des charges. Il couvre:

- Architecture des composants
- Étapes d'implémentation par phase
- Dépendances entre les modules
- Procédures de test
- Instructions de déploiement

---

2. Architecture Générale

2.1 Diagramme Flux Données

```
User Login
    ↓
Session Django (auth.User)
    ↓
Home Dashboard ← StatisticalService API
    ↓
Tableau des Notes
    ├─ Filter API (/api/filieres/, /api/niveaux/)
    ├─ Data API (/api/notes/)
    ├─ Edit Modal (PUT /api/notes/update/)
    ├─ Import (/api/notes/import/) → ExcelService
    └─ Export (/api/notes/export/) → ExcelService
```

2.2 Stack Layer Diagram

```
PRESENTATION LAYER (Templates HTML + JS)
├── base_adminlte.html (Master template)
├── home_adminlte.html (Dashboard)
├── tableau_notes_adminlte.html (Notes table)
└── grades.js (AJAX controller)

BUSINESS LOGIC LAYER (Views + Services)
├── views.py
│   ├── home() → GET /
│   ├── tableau_notes() → GET /tableau/
│   ├── notes_json() → GET /api/notes/
│   ├── filieres_json() → GET /api/filieres/
│   ├── niveaux_json() → GET /api/niveaux/
```

```
│   ├── import_notes() → POST /api/notes/import/
│   └── export_notes() → GET /api/notes/export/
├── services/excel_service.py (ExcelService class)
│   ├── import_from_excel()
│   └── export_to_excel()
└── mixins.py (LoginRequired, PermissionRequired)

DATA LAYER (Models + ORM)
├── Département
├── Filière
├── Niveau
├── UE
├── Étudiant (User FK)
└── Note
```

---

3. Phase 1: Foundation (BASE) - ✅ COMPLÉTÉE

Tâches complétées

3.1.1 Setup Initial

- ☑ Initialiser projet Django
- ☑ Créer app `notes`
- ☑ Configurer settings.py
- ☑ Créer models.py avec tous modèles
- ☑ Configuration admin.py
- ☑ Migrations initiales (0001_initial, 0002_ue_note)

3.1.2 Template Base (AdminLTE)

- ☑ Template `base_adminlte.html` (494 lignes)
  - Sticky navbar avec Home/Notes/Admin links
  - Username affichage + Logout séparé
  - Flexbox layout pour footer fixe
  - CDN avec SRI hashes (9 ressources)
  - Badge styling global
- ☑ Template inherit dans tous les enfants

3.1.3 Authentification

- ☑ Django auth par défaut (login_required decorator)
- ☑ Intégration navbar avec {{ user.username }}
- ☑ Logout button href="{% url 'logout' %}"
- ☑ Login page (si déjà existe, template héritant de base_adminlte)

3.1.4 Dashboard Couche Métier

- ☑ View `home()` avec context data:

```
context = {
    'total_students': Etudiant.objects.count(),
    'total_filieres': Filiere.objects.count(),
    'total_ues': UE.objects.count(),
    'average_note': Note.objects.aggregate(Avg('moyenne'))['moyenne__avg'],
    'recent_notes': Note.objects.order_by('-id')[:5]
}
```

- ☑ Template `home_adminlte.html` avec 6 tuiles stat
- ☑ Actions rapides (liens)
- ☑ Notes récentes affichées

3.1.5 Tests Unitaires

- ☑ 12 tests dans `tests.py`:
  - Auth login/logout ✓
  - Home view ✓
  - Admin Django ✓
  - CRUD models ✓
- ☑ Exécution: `python manage.py test` ✓ 12/12 passing

3.1.6 Git Setup

- ☑ `.gitignore` (Django standard)
- ☑ `git init` + 2 commits
- ☑ `README.md` avec instructions
- ☑ `requirements.txt` avec dépendances

État Phase 1: ✅ **COMPLÉTÉE 100%**

---

4. Phase 2: Tableau des Notes - ✅ COMPLÉTÉE

Tâches complétées

4.1 API REST Endpoints

4.1.1 GET /api/filieres/

```
# views.py
def filieres_json(request):
    """Retourne filieres filtrées par département en JSON"""
    departement_id = request.GET.get('departement')

    Filieres = Filiere.objects.all()
    if departement_id:
        filieres = filieres.filter(departement_id=departement_id)
```

```
    return JsonResponse({
        'filieres': list(filieres.values('id', 'nom'))
    })
```

- ☑ Filtre par département
- ☑ Retour JSON
- ☑ Intégration grades.js

4.1.2 GET /api/niveaux/

```
def niveaux_json(request):
    """Retourne niveaux filtrés par filière"""
    filiere_id = request.GET.get('filiere')

    niveaux = Niveau.objects.all()
    if filiere_id:
        niveaux = niveaux.filter(filiere_id=filiere_id)

    return JsonResponse({
        'niveaux': list(niveaux.values('id', 'numero'))
    })
```

- ☑ Filtre par filière
- ☑ Retour JSON
- ☑ Intégration grades.js

4.1.3 GET /api/notes/

```
def notes_json(request):
    """Retourne notes avec filtres department/filiere/niveau"""
    dep_id = request.GET.get('departement')
    fil_id = request.GET.get('filiere')
    niv_id = request.GET.get('niveau')

    # Filtrer UEs et étudiants
    ues_qs = UE.objects.all()
    students_qs = Etudiant.objects.all()

    if dep_id:
        ues_qs = ues_qs.filter(filiere__departement_id=dep_id)
        students_qs = students_qs.filter(filiere__departement_id=dep_id)
    if fil_id:
        ues_qs = ues_qs.filter(filiere_id=fil_id)
        students_qs = students_qs.filter(filiere_id=fil_id)
    if niv_id:
        ues_qs = ues_qs.filter(filiere__niveau_id=niv_id)
        students_qs = students_qs.filter(filiere__niveau_id=niv_id)
```

```
    # Construire tableau
    data = {
        'niveaus': [...],
        'students': [...],
        'notes': [...]
    }
    return JsonResponse(data)
```

- ☑ Filtres multiples (département, filière, niveau)
- ☑ Retour complètement structuré
- ☑ Pagination (TBD côté frontend)

4.1.4 POST /api/notes/import/

```
def import_notes(request):
    """Importe notes depuis Excel"""
    if request.method == 'POST':
        file = request.FILES.get('file')
        ue_id = request.POST.get('ue_id')

        # ExcelService.import_from_excel()
        # Validation + insertion

        return JsonResponse({'status': 'success'})
```

- ☑ Upload fichier Excel
- ☑ Sélection UE
- ☑ Validation données
- ☑ Insertion DB

4.1.5 GET /api/notes/export/

```
def export_notes(request):
    """Exporte notes en Excel"""
    filiere_id = request.GET.get('filiere')
    niveau_id = request.GET.get('niveau')
    ue_id = request.GET.get('ue_id')

    # ExcelService.export_to_excel()
    # Construcción du fichier

    response = HttpResponse(content_type='...')
    response['Content-Disposition'] = 'attachment; filename="notes.xlsx"'
    return response
```

- ☑ Export filtré

- ☑ Sélection UE
- ☑ Retour fichier téléchargeable

4.2 Frontend - grades.js (381 lignes)

4.2.1 Gestion des Filtres

```
// Filter change handlers
document.getElementById('filter-departement').addEventListener('change', async
(event) => {
    const depId = event.target.value;

    // Update filière dropdown via GET /api/filieres/?departement=X
    const response = await fetch(`/api/filieres/?departement=${depId}`);
    const data = await response.json();
    // Populate options...

    updateImportExportButtons(); // Disable until niveau selected
    loadTable(); // Reload avec nouveau département
});
```

- ☑ Cascading: département → filière → niveau
- ☑ API async calls (fetch)
- ☑ Désactivation boutons import/export
- ☑ Table refresh auto

4.2.2 Édition Notes

```
// Modal edit note
function openEditModal(studentId, ueId) {
    // GET /api/notes/?etudiant=X&ue=Y
    // Populate modal fields (CC, TP, SN)
    // Save via PATCH /api/notes/{id}/
}
```

- ☑ Modal avec champs CC/TP/SN
- ☑ Validation (0-20)
- ☑ Permission check (editable attribute)
- ☑ Mise à jour table post-save

4.2.3 Pagination

```
// Page navigation
const totalPages = Math.ceil(totalStudents / pageSize);
document.getElementById('prev-page').addEventListener('click', () => {
    currentPage--;
```

```
    loadTable();
});
```

- ☑ Boutons Prev/Next
- ☑ Affichage page courante
- ☑ Recharge table

4.2.4 Import/Export

```
// Export notes
document.getElementById('export-notes-btn').addEventListener('click', () => {
    const filiere = document.getElementById('filter-filiere').value;
    const niveau = document.getElementById('filter-niveau').value;
    const ue = document.getElementById('filter-ue').value;

    window.location.href = `/api/notes/export/?
filiere=${filiere}&niveau=${niveau}&ue=${ue}`;
});

// Import notes
document.getElementById('#import-form').addEventListener('submit', async (e) => {
    e.preventDefault();
    const formData = new FormData(e.target);

    const response = await fetch('/api/notes/import/', {
        method: 'POST',
        body: formData
    });

    if (response.ok) {
        loadTable(); // Refresh après import
    }
});
```

- ☑ Export download direct
- ☑ Import FormData POST
- ☑ Boutons désactivés until niveau selected
- ☑ Error handling

4.3 Template - tableau_notes_adminlte.html (173 lignes)

```
<!-- Filter section -->
<select id="filter-departement">
    <option>-- Département --</option>
</select>
<select id="filter-filiere" disabled>
    <option>-- Filière --</option>
</select>
```

```
<select id="filter-niveau" disabled>
    <option>-- Niveau --</option>
</select>

<!-- Buttons -->
<button id="export-notes-btn" disabled>Export Excel</button>
<button id="import-notes-btn" disabled>Import Excel</button>

<!-- Table -->
<table id="notes-table">
    <!-- Rows générés par JavaScript -->
</table>

<!-- Modals pour édition + import -->
<div id="editModal">...</div>
<div id="importModal">...</div>
```

- ☑ Filter dropdowns
- ☑ Buttons (avec disabled state)
- ☑ Notes table structure
- ☑ Edit modal
- ☑ Import modal

4.4 Services (ExcelService) - TBD

```
# services/excel_service.py

class ExcelService:

    @staticmethod
    def export_to_excel(filiere, niveau, ues, students):
        """Exporte notes en workbook Excel"""
        wb = Workbook()
        ws = wb.active

        # Header
        ws['A1'] = 'Étudiant'
        # ... colonnes UE

        # Rows
        for idx, student in enumerate(students, start=2):
            ws[f'A{idx}'] = student.nom
            # ... notes

        return wb

    @staticmethod
    def import_from_excel(file_path, ue_id):
        """Importe notes depuis fichier Excel"""
        wb = load_workbook(file_path)
```

```
        ws = wb.active

        notes_created = []
        for row in ws.iter_rows(min_row=2):
            student_name = row[0].value
            cc = row[1].value
            tp = row[2].value
            sn = row[3].value

            # Validation
            # Création Note

        return notes_created
```

État Phase 2: ✅ **COMPLÉTÉE 100%**

---

5. Phase 3: Améliorations UX (EN COURS)

Tâches à compléter

5.1 Graphiques de Performance

Objectif: Ajouter visualisation ChartJS sur tableau notes

Fichers à créer:

- `static/js/charts.js` - Initialisation Chart.js
- Update `tableau_notes_adminlte.html` - Ajouter div pour graphique

Étapes:

1. Installer Chart.js via CDN
2. Créer fonction `initChart()` dans grades.js
3. Après loadTable(), appeler initChart()
4. Afficher:
    - Histogramme moyennes par UE
    - Pie chart répartition notes (< 10, 10-15, > 15)

```
// Pseudo-code: grades.js
function initChart() {
    const ctx = document.getElementById('statsChart').getContext('2d');
    new Chart(ctx, {
        type: 'bar',
        data: {
            labels: ues.map(u => u.nom),
            datasets: [{
                label: 'Moyenne',
                data: ues.map(u => u.moyenne),
                backgroundColor: 'rgba(75, 192, 192, 0.6)'
            }]
```

```
        }
    });
}
```

Dépendances: Chart.js 3.9.1+ (CDN)
Effort estimé: 4-6 heures

5.2 Recherche par Nom d'Étudiant

Objectif: Filtrer étudiants par prénom/nom en temps réel

Fichiers à modifier:

- `static/js/grades.js` - Ajouter filtrage côté client
- `tableau_notes_adminlte.html` - Input recherche

Étapes:

1. Ajouter input `<input id="search-student">`
2. Dans grades.js: listener sur input
3. Re-filtrer tabledata localement (sans API call)
4. Mettre à jour affichage table

```
document.getElementById('search-student').addEventListener('keyup', (e) => {
    const searchTerm = e.target.value.toLowerCase();
    const rows = document.querySelectorAll('#notes-table tbody tr');

    rows.forEach(row => {
        const studentName = row.cells[0].textContent.toLowerCase();
        row.style.display = studentName.includes(searchTerm) ? '' : 'none';
    });
});
```

Dépendances: Aucune
Effort estimé: 2-3 heures

5.3 Validations Avancées Import/Export

Objectif: Ajouter validations côté serveur et client avant import destructif

Fichiers à modifier:

- `views.py` - Fonctions import_notes(), export_notes()
- `static/js/grades.js` - Validations côté client
- `tableau_notes_adminlte.html` - Messages feedback

Étapes:

1. **Côté client**: Avant import, vérifier:
    - Fichier Excel valide (extension)
    - Colonne UE existe
    - Données numériques 0-20
2. **Côté serveur** (views.py):
    - Validation stricte avec try/except
    - Feedback d'erreur JSON
    - Transaction rollback si erreur
3. **Modal confirmation**:
    - Afficher "Importer X notes?" avant POST
    - Bouton Cancel/Import

Pseudo-code:

```
# views.py - import_notes()
try:
    wb = load_workbook(file)
    errors = []

    for row in wb.active.iter_rows():
        if not is_valid_note(row[1].value):
            errors.append(f"Ligne {row[0]}: note invalide")

    if errors:
        return JsonResponse({'status': 'error', 'errors': errors}, status=400)

    # Proceed with import

except Exception as e:
    return JsonResponse({'status': 'error', 'message': str(e)}, status=500)
```

Dépendances: openpyxl (déjà installé)
Effort estimé: 3-4 heures

5.4 Export PDF

Objectif: Générer rapport PDF formaté des notes

Fichiers à créer:

- `services/pdf_service.py` - PDFService class
- Update `tableau_notes_adminlte.html` - Bouton Export PDF
- Update `views.py` - Endpoint export_pdf

Étapes:

1. Créer PDFService avec reportlab
2. Ajouter bouton "Export PDF" (similaire à Excel)
3. Endpoint POST /api/notes/export-pdf/
4. Générer tableau formaté avec headers

```
# services/pdf_service.py

class PDFService:
    @staticmethod
    def generate_report(filiere, niveau, students, notes):
        """Génère PDF des notes"""
        buffer = BytesIO()
        doc = SimpleDocTemplate(buffer, pagesize=landscape(A4))

        # Create table data
        data = [['Étudiant', 'UE1', 'UE2', '...']]
        for student in students:
            row = [student.nom]
            # ... ajouter notes
            data.append(row)

        # Create table
        table = Table(data)
        table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            # ... plus de styles
        ]))

        # Build
        elements = [Paragraph(f'Notes {filiere.nom}'), table]
        doc.build(elements)

        buffer.seek(0)
        return buffer
```

Dépendances: reportlab (déjà installé)
Effort estimé: 4-5 heures

5.5 Historique Modifications

Objectif: Logger toutes modifications de notes avec timestamp et auteur

Fichiers à créer:

- `models.py` - Model NoteHistory
- `migrations/` - Migration pour NoteHistory
- `services/audit_service.py` - AuditService

Étapes:

1. Créer model:

```
class NoteHistory(models.Model):
    note = models.ForeignKey(Note, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    old_cc = models.DecimalField(max_digits=5, decimal_places=2, null=True)
    new_cc = models.DecimalField(max_digits=5, decimal_places=2, null=True)
    old_tp = models.DecimalField(max_digits=5, decimal_places=2, null=True)
    new_tp = models.DecimalField(max_digits=5, decimal_places=2, null=True)
    old_sn = models.DecimalField(max_digits=5, decimal_places=2, null=True)
    new_sn = models.DecimalField(max_digits=5, decimal_places=2, null=True)
    timestamp = models.DateTimeField(auto_now_add=True)
```

2. Dans views.py - import_notes():

```
@audit_required
def import_notes(request):
    # ... import logic
    for note in notes_created:
        AuditService.log_change(
            note=note,
            user=request.user,
            old_values=None,  # C'était nouveau
            new_values={'cc': note.cc, 'tp': note.tp, 'sn': note.sn}
        )
```

3. Admin interface pour voir historique

Dépendances: Django ORM
Effort estimé: 5-6 heures

Roadmap Phase 3

```
Semaine 1:
- Jour 1-2: Graphiques Chart.js
- Jour 3: Recherche étudiant
- Jour 4: Code review

Semaine 2:
- Jour 1-2: Validations avancées
- Jour 3: Export PDF
- Jour 4: Tests

Semaine 3:
- Jour 1-2: Historique modifications
```

```
- Jour 3: Documentation
- Jour 4: Integration testing
```

---

6. Phase 4: Production (PLANIFIÉE)

6.1 Déploiement

Option A: PythonAnywhere (Recommandé pour débutants)

1. Créer compte PythonAnywhere
2. Upload code via Git:

```
git clone https://github.com/T4zor/Projet-Django-Gestion-de-notes.git
cd Projet-Django-Gestion-de-notes
pip install -r requirements.txt
```

3. Configurer Web App:
   - Framework: Django 4.2
   - Python 3.8+
   - WSGI: /path/to/wsgi.py
4. Configurer BD:
   - `python manage.py migrate`
   - Upload data (ou import depuis local)
5. Static Files:
   - `python manage.py collectstatic`
   - Configure dans Web app settings

Option B: Heroku (Alternative gratuit/payant)

```
# Install Heroku CLI
heroku login

# Create app
heroku create notre-app

# Configure Procfile
echo "web: gunicorn backend.wsgi" > Procfile

# Deploy
git push heroku main

# Migrate
heroku run python manage.py migrate
```

Option C: VPS Autohébergé (Contrôle total)

- DigitalOcean / Linode / Vultr
- Nginx + Gunicorn
- Systemd service pour Django
- PostgreSQL

6.2 Configuration Sécurité

```
# settings.py production

ALLOWED_HOSTS = ['votredomaine.com', 'www.votredomaine.com']

SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True

DEBUG = False

# Database
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'notes_db',
        'USER': 'notes_user',
        'PASSWORD': os.environ.get('DB_PASSWORD'),
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

# Email pour reset password
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = os.environ.get('EMAIL_USER')
EMAIL_HOST_PASSWORD = os.environ.get('EMAIL_PASSWORD')
```

6.3 Maintenance

- Backup hebdomadaires (pg_dump)
- Monitoring logs (tail -f gunicorn.log)
- Updates Django/packages (quarterly)
- Health checks (uptime robot)

---

7. Procédures de Test

7.1 Tests Unitaires (Django TestCase)

Exécuter:

```
python manage.py test notes
# Attendu: OK (12 tests)
```

Tester nouvelles fonctionnalités:

```
# tests.py

class ChartViewTest(TestCase):
    def test_chart_api_returns_valid_json(self):
        response = self.client.get('/api/stats/chart/')
        self.assertEqual(response.status_code, 200)
        self.assertIsInstance(response.json(), dict)

class SearchTest(TestCase):
    def test_search_student_filters_table(self):
        # Create test student
        student = Etudiant.objects.create(nom='Dupont', prenom='Jean')
        # Test filtering
        response = self.client.get('/tableau/notes/?search=Dupont')
        self.assertContains(response, 'Dupont')
```

7.2 Tests Intégration

```
# Scénario complet: Login → Filter → Edit → Export
1. Login
2. Navigate to /tableau/notes/
3. Select Département, Filière, Niveau
4. Edit une note (CC=15)
5. Click Export
6. Vérifier fichier Excel contains CC=15
```

7.3 Tests Acceptance (Manuel)

Sprint Checklist:

- ☐ Filtres cascadent correctement
- ☐ Notes éditables according to permissions
- ☐ Import/Export marche
- ☐ No JavaScript console errors
- ☐ Responsive design (mobile/tablet)
- ☐ Load time < 2s

---

8. Checklist Déploiement

Avant de mettre en production:

- [ ] All tests passing (12/12)
- [ ] No Django warnings/errors
- [ ] settings.DEBUG = False
- [ ] ALLOWED_HOSTS configured
- [ ] CSRF_TRUSTED_ORIGINS set
- [ ] SSL/HTTPS enabled
- [ ] Database backup plan
- [ ] Environment variables configured
- [ ] Static files collected
- [ ] Email configured
- [ ] Monitoring/logging active
- [ ] Disaster recovery plan

---

9. Problèmes Connus et TODOs

Bugs Connus

- [ ] AdminLTE sidebar remnants (CSS cleanup needed)
- [ ] Export très lent si > 1000 étudiants
- [ ] Mobile: table notes scrolling difficulty

Technical Debt

- [ ] Extract ExcelService to standalone class
- [ ] Add REST framework serializers
- [ ] Admin interface customization
- [ ] Add middleware for request logging

Optimisations Futures

- [ ] Redis cache pour filtres
- [ ] Elasticsearch pour search
- [ ] Async tasks (Celery) pour import large
- [ ] WebSocket pour real-time updates

---

10. Ressources et Références

Documentation

- Django Official Docs
- AdminLTE Components
- openpyxl Tutorial
- reportlab Usage

Tools

- Django Debug Toolbar: `pip install django-debug-toolbar`
- Django Extensions: `pip install django-extensions`

- Coverage.py: `pip install coverage`

Commandes Utiles

```
# Django management
python manage.py shell           # Python REPL avec Django
python manage.py makemigrations  # Créer migrations
python manage.py migrate         # Appliquer migrations
python manage.py createsuperuser # Créer admin user
python manage.py collectstatic   # Gather static files

# Testing
python manage.py test
coverage run --source='.' manage.py test
coverage report

# Database
python manage.py dumpdata > backup.json  # Export data
python manage.py loaddata backup.json    # Import data

# Git
git log --oneline              # Voir commits
git diff HEAD~1                # Compare
git revert <commit>            # Undo commit
```

---

11. sign-off et Approbations

Étape	Responsable	Date	Status
Phase 1 (Foundation)	T4zor	01/02/2026	✅ Complétée
Phase 2 (Tableau Notes)	T4zor	10/02/2026	✅ Complétée
Phase 3 (UX Improvements)	T4zor	TBP	⏳ Planifiée
Phase 4 (Production)	TBD	TBD	⏳ Planifiée

Document généré: 11/02/2026
Prochaine mise à jour: Après Phase 3
Version: 1.0