**COURS**

**Écrivez du code Python maintenable avec MVC et SOLID**

# P1C6 - Faites évoluer et documentez votre application

## Vérifiez votre travail

**La politique et son assemblage.** La politique est celle présentée plus haut. La racine de composition reçoit une ligne de plus, et rien d'autre :

```
def build_policy_resolver() -> LeavePolicyResolver:
    return LeavePolicyResolver(
        {
            ContractType.PERMANENT: PermanentContractPolicy(),
            ContractType.FIXED_TERM: FixedTermContractPolicy(
                days_per_full_month=settings.HAMAC_FIXED_TERM_DAYS_PER_FULL_MONTH,
            ),
            ContractType.APPRENTICESHIP: ApprenticeshipPolicy(),
            ContractType.INTERNSHIP: InternshipPolicy(),
        }
    )
```

N'oubliez pas d'importer `InternshipPolicy` en haut du module. Si vous l'oubliez, Ruff vous le signale avant même de lancer les tests : `F821 Undefined name \`InternshipPolicy\\`.

**Les tests.** Les deux tests unitaires reprennent les dates du test des CDD :

```
def test_internship_policy_counts_only_full_months():
    policy = InternshipPolicy()
    employee = Employee(hire_date=date(2026, 3, 15))

    entitlement = policy.compute_entitlement(employee, date(2026, 10, 20))

    assert entitlement == 14


def test_internship_policy_gives_no_days_before_the_internship_starts():
    policy = InternshipPolicy()
    employee = Employee(hire_date=date(2026, 10, 27))

    entitlement = policy.compute_entitlement(employee, date(2026, 10, 20))

    assert entitlement == 0
```

Le premier compte 7 mois complets, soit 14 jours. Le calcul ne porte ici que sur des nombres entiers et sur un taux de 2, sans arrondi : une égalité exacte suffit. Le second vérifie le cas de Zoé, sept jours avant son arrivée : sans `max(0, …)`, il échouerait avec `-2.0`.

Le test d'API vérifie le solde de Zoé dans les conditions réelles, avec le jeu de démonstration :

```
@pytest.mark.django_db
def test_balance_of_intern_is_zero_before_the_internship_starts(client, demo_data):
    response = client.get("/api/employees/6/balance")

    assert response.status_code == 200
    assert response.json()["balance"] == 0
```

**La configuration de Ruff.** Dans `pyproject.toml` :

```
[tool.ruff.lint]
select = ["E", "F", "I", "B", "UP", "DJ"]
```

Les avertissements `F401` et `I001` se corrigent avec `--fix`. Les deux `DJ008` demandent une méthode `__str__` sur chaque modèle :

```
class Employee(models.Model):
    # ...

    def __str__(self) -> str:
        return f"{self.first_name} {self.last_name}"


class LeaveRequest(models.Model):
    # ...

    def __str__(self) -> str:
        return f"Demande n° {self.pk} ({self.status})"
```

Ajouter `__str__` ne modifie pas la base : `makemigrations` ne génère aucune migration.

**Le README.** Voici une version qui répond aux questions de Tomás sans recopier le code :

```
# Hamac

API interne de gestion des congés : les salariés consultent leur solde
et déposent leurs demandes, les managers les approuvent ou les refusent.
L'écran est développé par l'équipe front, dans un autre dépôt.

## Démarrer

Prérequis : uv (il installe lui-même la version de Python du projet).

    uv sync
    uv run python manage.py migrate
    uv run python manage.py reset_demo   # charge le jeu de démonstration
    uv run python manage.py runserver    # API sur http://localhost:8000

La base SQLite (db.sqlite3) n'est pas versionnée. Relancez reset_demo pour
retrouver l'état initial des données de démonstration.

## Configuration

| Réglage | Où | Rôle |
| --- | --- | --- |
| HAMAC_FIXED_TERM_DAYS_PER_FULL_MONTH | Variable d'environnement (défaut : 2.08) | Taux
    d'acquisition des CDD |
| DEBUG, DATABASES | config/settings.py | Mode de développement, base de données |

## Architecture

Projet Django config, application unique leave :

- models.py : Employee, LeaveRequest, LeaveStatus
- policies.py : une politique de calcul des droits par type de contrat
- services.py : LeaveService, qui porte les règles métier
- composition.py : seul module qui assemble les objets
- serializers.py, views.py, urls.py : l'API (DRF), sans règle métier

Ajouter un type de contrat : une valeur dans ContractType, une classe de
politique dans policies.py, une ligne dans composition.py, des tests.
Le service ne change pas.

## API

Documentation interactive : http://localhost:8000/api/docs (en développement).
Contrat versionné : schema.yml (régénérer avec
```

```
uv run python manage.py spectacular --file schema.yml).

| Verbe | Route | Rôle |
| --- | --- | --- |
| GET | api/employees/{id}/balance | Solde d'un salarié |
| GET | api/leave-requests?status= | Liste des demandes, filtrable par statut |
| GET | api/leave-requests/{id} | Détail d'une demande |
| POST | api/leave-requests | Dépôt d'une demande (201) |
| POST | api/leave-requests/{id}/approval | Approbation (409 si déjà traitée) |
| POST | api/leave-requests/{id}/refusal | Refus (409 si déjà traitée) |

Les clés JSON sont en camelCase : elles font partie du contrat.

## Tests et qualité

    uv run pytest
    uv run ruff check .
    uv run ruff format --check .

## Règles métier et limites connues

Les règles de calcul des droits sont validées par Samuel Okafor (RH) :
ne modifiez pas une règle sans son accord.

- Les jours fériés ne sont pas déduits du nombre de jours ouvrés.
- L'API n'a pas d'authentification : n'importe quel client peut approuver une demande.
- CORS n'est pas configuré : à ajouter (django-cors-headers) quand l'origine de l'écran
    sera connue.
- Les messages d'erreur de DRF et de Hamac sont en français, conformément à la
    configuration du projet.
- La politique des CDD renvoie un droit négatif pour une date d'entrée future (à confirmer
     avec Samuel).
- Configuration de développement uniquement : DEBUG, SECRET_KEY et SQLite sont à revoir
    avant une mise en production.
```

La distinction attendue est la suivante : une application qui **fonctionne** tourne sur le poste de la personne qui l'a écrite ; une application **livrée** peut être lancée, comprise et modifiée par quelqu'un d'autre, sans l'aide de son auteur. Le README, les tests, le linter et le contrat versionné sont ce qui fait passer Hamac de l'une à l'autre.

## Erreurs fréquentes

• **Modifier** `LeaveService` **pour gérer les stages** : le résolveur et la racine de composition rendent cette modification inutile.

• **Oublier** `max(0, …)` **dans** `InternshipPolicy` : le solde de Zoé vaut -2 jours, et le test du stage qui n'a pas commencé échoue.

• **Supprimer la vérification du résolveur** parce qu'elle fait échouer les tests : elle fait exactement son travail, en signalant la politique manquante.

• **Formater le code et corriger un bug dans le même commit** : la correction se perd au milieu des changements de mise en page.

• **Faire taire Ruff avec des** `# noqa` **sans explication**, ou en retirant une famille de règles : les avertissements disparaissent, mais les problèmes restent.

• **Envoyer** `schema.yml` **sans le versionner** : au prochain changement, l'équipe front travaillera sur un contrat périmé, sans que personne ne s'en aperçoive.

• **Vérifier le README sur votre propre poste** : votre base, votre environnement et vos habitudes cachent les étapes manquantes. Seul un clone neuf les révèle.