COURS

Écrivez du code Python maintenable avec MVC et SOLID

P1C5 - Exposez vos fonctionnalités avec Django REST Framework

Vérifiez votre travail

Les statuts. Dans `leave/models.py`, les statuts rejoignent une énumération, comme les types de contrat au chapitre 3 :

```
class LeaveStatus(models.TextChoices):
    PENDING = "Pending"
    APPROVED = "Approved"
    REFUSED = "Refused"
```

Le champ devient `models.CharField(max_length=20, choices=LeaveStatus, default=LeaveStatus.PENDING)`. Comme une `StrEnum`, chaque membre est une chaîne : la base stocke toujours `"Pending"`. `LeaveStatus.choices` alimente le `ChoiceField` du filtre. La migration ne modifie pas la table : Django enregistre seulement la nouvelle description du champ.

Le service. `ApprovalResult` gagne un cas `NOT_PENDING`, et une énumération `RefusalResult` (`REFUSED`, `NOT_FOUND`, `NOT_PENDING`) accompagne la nouvelle méthode `refuse`, construite sur le même modèle que `approve`. Voici les principales méthodes nouvelles ou modifiées :

```
def get_requests(self, status: str | None = None) -> QuerySet[LeaveRequest]:
    leave_requests = LeaveRequest.objects.order_by("start_date")
    if status is not None:
        leave_requests = leave_requests.filter(status=status)
    return leave_requests

def create(self, employee_id: int, start_date: date, end_date: date) -> LeaveRequest |
    None:
    if not Employee.objects.filter(pk=employee_id).exists():
        return None
    return LeaveRequest.objects.create(
        employee_id=employee_id,
        start_date=start_date,
        end_date=end_date,
        working_days=count_working_days(start_date, end_date),
    )

@transaction.atomic
def approve(self, request_id: int) -> ApprovalResult:
    leave_request = (
        LeaveRequest.objects.select_related("employee").filter(pk=request_id).first()
    )
    if leave_request is None:
        return ApprovalResult.NOT_FOUND
    if leave_request.status != LeaveStatus.PENDING:
        return ApprovalResult.NOT_PENDING
    # ... la suite ne change pas
```

`get_request` renvoie `LeaveRequest.objects.filter(pk=request_id).first()`. `count_working_days` est une fonction du module, qui compte les jours dont `weekday()` est inférieur à `5`, du lundi au vendredi. Elle ignore les jours fériés : dans un vrai projet, ce serait une question à poser à Samuel. La vérification du statut se trouve dans le service, et non dans la vue : elle s'applique quel que soit le client.

Les vues. Le module `leave/views.py` est réécrit. Voici l'essentiel du `ViewSet` ; `list`, `retrieve`, `refusal` et `employee_balance` suivent le même modèle :

```
class LeaveRequestViewSet(viewsets.ViewSet):
    lookup_value_regex = r"\d+"

    @extend_schema(request=CreateLeaveRequestSerializer, responses={201:
        LeaveRequestSerializer})
    def create(self, request):
        payload = CreateLeaveRequestSerializer(data=request.data)
        payload.is_valid(raise_exception=True)
        leave_request = build_leave_service().create(**payload.validated_data)
        if leave_request is None:
            raise ValidationError({"employeeId": ["Ce salarié n'existe pas."]})
        return Response(
            LeaveRequestSerializer(leave_request).data,
            status=status.HTTP_201_CREATED,
            headers={"Location": reverse("leave-request-detail",
                args=[leave_request.pk])},
        )

    @extend_schema(request=None, responses=LeaveRequestSerializer)
    @action(detail=True, methods=["post"])
    def approval(self, request, pk):
        service = build_leave_service()
        match service.approve(int(pk)):
            case ApprovalResult.NOT_FOUND:
                raise NotFound()
            case ApprovalResult.NOT_PENDING:
                raise AlreadyProcessed()
            case ApprovalResult.APPROVED | ApprovalResult.INSUFFICIENT_BALANCE:
                return Response(LeaveRequestSerializer(service.get_request(int(pk))).data)
            case unreachable:
                assert_never(unreachable)
```

`@action(detail=True)` génère la route `leave-requests/{pk}/approval`, du nom de la méthode. `**payload.validated_data` fonctionne parce que les clés sont les noms de `source`, qui sont aussi les paramètres du service. `reverse` construit l'en-tête `Location` à partir du nom de la route. Les vues restent fidèles à la règle du chapitre 3 : elles valident la requête, appellent le service et traduisent son résultat en réponse HTTP.

Les tests. Les tests existants utilisent les nouvelles routes, par exemple `client.get("/api/employees/4/balance")` pour le bug de Leïla et le contrat du solde, et `client.post("/api/leave-requests/1/approval")` pour l'approbation. Les deux nouveaux tests vérifient le contrat :

```
@pytest.mark.django_db
def test_approving_a_request_twice_returns_conflict(client, demo_data):
    client.post("/api/leave-requests/7/approval")
    response = client.post("/api/leave-requests/7/approval")

    assert response.status_code == 409


@pytest.mark.django_db
def test_creating_a_request_ending_before_it_starts_returns_bad_request(client,
    demo_data):
    response = client.post(
        "/api/leave-requests",
        {"employeeId": 4, "startDate": "2026-07-10", "endDate": "2026-07-06"},
        content_type="application/json",
    )
```

```
    assert response.status_code == 400
    assert "endDate" in response.json()
```

Sans `content_type="application/json"`, le client de test enverrait un formulaire. Le premier test rejoue exactement l'incident de vendredi : s'il avait existé plus tôt, Amina n'aurait pas perdu cinq jours. Les tests du service et des politiques n'ont pas changé : le contrat de l'API a évolué, pas les règles RH.

Erreurs fréquentes

• **Migrer vers DRF et changer le contrat dans le même commit** : si un test échoue, vous ne savez plus si la cause est l'outil ou le contrat.

• **Laisser les clés JSON passer en snake_case** : l'API fonctionne, mais l'écran de l'équipe front ne trouve plus ses données.

• **Utiliser un** `ModelViewSet` **ou** `fields = "__all__"` : le client peut choisir le statut de sa demande sans passer par le service.

• **Vérifier le statut de la demande dans la vue** : la règle ne s'applique pas aux autres appelants du service.