P1C6 - Connectez les pages web fournies et documentez votre application

Vérifiez votre travail

Le template. Une ligne s'ajoute dans le `<head>` de `leave-requests.html` :

```
<meta name="api-base" th:content="@{/api/}">
```

leave-api.js. Chaque fonction correspond à une route, et toutes partagent la même gestion des erreurs :

```
import { LeaveApiError } from './leave-api-error.js';

const baseUrl = document.querySelector('meta[name="api-base"]').content;

export async function getBalance(employeeId) {
  const balance = await send(`employees/${employeeId}/balance`);
  return balance.balance;
}

export async function getRequests(status) {
  const query = status ? `?status=${encodeURIComponent(status)}` : '';
  const requests = await send(`leave-requests${query}`);
  return requests.map(toLeaveRequest);
}

export async function createRequest(newRequest) {
  const created = await send('leave-requests', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(newRequest),
  });
  return toLeaveRequest(created);
}

export async function approve(requestId) {
  return toLeaveRequest(await send(`leave-requests/${requestId}/approval`, { method: 'POST' }));
}

export async function refuse(requestId) {
  return toLeaveRequest(await send(`leave-requests/${requestId}/refusal`, { method: 'POST' }));
}

function toLeaveRequest(dto) {
  return {
    id: dto.id,
    employeeId: dto.employeeId,
    startDate: dto.startDate,
    endDate: dto.endDate,
    days: dto.workingDays,
    status: dto.status,
  };
}

async function send(path, options = {}) {
  const response = await fetch(baseUrl + path, options);
  if (response.ok) {
    return response.json();
```

```
  }
  throw new LeaveApiError(await readErrorMessage(response));
}

async function readErrorMessage(response) {
  const fallback = `Erreur ${response.status}`;
  const contentType = response.headers.get('Content-Type') ?? '';
  if (!contentType.includes('application/problem+json')) {
    return fallback;
  }
  const problem = await response.json();
  return Object.values(problem.errors ?? {})[0] ?? problem.detail ?? fallback;
}
```

`toLeaveRequest` liste explicitement les champs, plutôt que de copier tout l'objet reçu : on voit d'un coup d'œil quel champ du contrat devient quelle propriété de la page, et un champ ajouté un jour à l'API n'apparaît pas dans la page sans que personne ne l'ait décidé. `encodeURIComponent` protège la chaîne de requête si un statut contenait un jour un caractère spécial. La vérification du type de contenu protège contre une réponse d'erreur qui ne serait pas du JSON, par exemple une page HTML renvoyée par un proxy. Seules les fonctions exportées sont visibles de `app.js` : `send`, `readErrorMessage` et `toLeaveRequest` restent privées au module, comme une méthode `private` en Java.

app.js. Une seule ligne change :

```
import * as leaveApi from './leave-api.js';
```

Conservez `leave-api-mock.js` dans le projet : il permettra encore de travailler sur la page sans dépendre des données de l'API.

La politique des stages. `Math.max` garantit le contrat commun des politiques, même avant l'arrivée de la personne :

```
@Component
final class InternshipPolicy implements LeaveEntitlementPolicy {

    private static final long DAYS_PER_MONTH = 30;
    private static final double DAYS_PER_FULL_MONTH = 2;

    @Override
    public boolean supports(ContractType type) {
        return type == ContractType.INTERNSHIP;
    }

    @Override
    public double computeEntitlement(Employee employee, LocalDate today) {
        var daysSinceHire = ChronoUnit.DAYS.between(employee.getHireDate(), today);
        var fullMonths = Math.floorDiv(daysSinceHire, DAYS_PER_MONTH);
        return Math.max(0, fullMonths) * DAYS_PER_FULL_MONTH;
    }
}
```

Aucune autre ligne de code n'a changé, en dehors de la constante de l'enum : le scan des composants trouve la classe, et le résolveur la reçoit dans sa liste.

Les tests. Les deux tests unitaires reprennent les dates du test de la politique des CDD, pour faciliter la comparaison. Ils n'ont besoin ni de Spring ni de la base :

```
class InternshipPolicyTests {

    private final InternshipPolicy policy = new InternshipPolicy();
```

```
    @Test
    void grantsTwoDaysPerFullMonth() {
        var employee = new Employee();
        employee.setHireDate(LocalDate.of(2026, 3, 15));

        var entitlement = policy.computeEntitlement(employee, LocalDate.of(2026, 10, 20));

        assertThat(entitlement).isEqualTo(14.0);
    }

    @Test
    void grantsNothingBeforeInternshipStarts() {
        var employee = new Employee();
        employee.setHireDate(LocalDate.of(2026, 11, 2));

        var entitlement = policy.computeEntitlement(employee, LocalDate.of(2026, 10, 26));

        assertThat(entitlement).isEqualTo(0.0);
    }
}
```

Entre le 15 mars et le 20 octobre, il s'écoule 219 jours, soit 7 mois complets de 30 jours, et donc 14 jours de droits. Ici, une égalité exacte convient : `7 * 2` se calcule sans erreur d'arrondi. Le test est placé dans le même package que `InternshipPolicy`, sous `src/test/java`, pour pouvoir utiliser cette classe package-private.

Le test d'API vérifie que toute la chaîne fonctionne, du scan des composants jusqu'au DTO. Ajoutez-le à `LeaveApprovalTests`. Il échoue avec un `500` tant que l'enum ne connaît pas les stages, puis ne peut même plus démarrer le contexte de test tant que la politique manque :

```
@Test
void balanceOfInternIsAvailableBeforeArrival() throws Exception {
    var result = mvc.get().uri("/api/employees/6/balance").exchange();

    assertThat(result).hasStatusOk();
    double balance = JsonPath.read(result.getResponse().getContentAsString(), "$.balance");
    assertThat(balance).isEqualTo(0.0);
}
```

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

```
# Hamac

Application interne de gestion des congés : les salariés consultent leur solde
et déposent leurs demandes, les managers les approuvent ou les refusent.

## Démarrer

Prérequis : JDK 25. Maven n'a pas besoin d'être installé (wrapper fourni).

    ./mvnw spring-boot:run      # Windows : mvnw.cmd spring-boot:run

Application sur http://localhost:8080 (pages web et API).
La base H2 est en mémoire : elle est recréée à chaque démarrage à partir de
src/main/resources/data.sql.

## Configuration

Fichier : src/main/resources/application.properties

| Propriété | Rôle |
| --- | --- |
| hamac.leave.fixed-term-days-per-full-month | Droits des CDD par mois complet (2.08) |
| logging.level.org.hibernate.SQL | Affichage des requêtes SQL (debug en développement) |
```

```
Toute propriété peut être remplacée par une variable d'environnement, par exemple
HAMAC_LEAVE_FIXEDTERMDAYSPERFULLMONTH=2.2.

## Architecture

Packages par fonctionnalité, sous com.hamac :

- employee : entité Employee, type de contrat, repository.
- leave : contrôleurs REST, LeaveService (règles métier), politiques de congés
  (une classe par type de contrat), DTO, repository des demandes.
- notification : envoi des notifications.
- web : contrôleur de la page, gestion des erreurs de l'API (ProblemDetail).
- Pages : src/main/resources/templates (Thymeleaf) et static/js.
  leave-api.js appelle l'API, leave-api-mock.js fournit des données simulées.

Ajouter un type de contrat : une constante dans ContractType, une classe
@Component qui implémente LeaveEntitlementPolicy, un test unitaire.
L'application refuse de démarrer si un type de contrat n'a pas de politique.

## API

Contrat complet et essais : http://localhost:8080/swagger-ui.html
Description OpenAPI : http://localhost:8080/v3/api-docs

| 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 erreurs sont renvoyées au format ProblemDetail (RFC 9457).

## Tests

    ./mvnw test

Tests unitaires des politiques de congés, tests d'API du parcours principal
(solde, approbation, double approbation, validation).

## 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.
- Les messages d'erreur de l'API sont en français, quelle que soit la langue de la page.
- La politique des CDD renvoie un droit négatif pour une date d'entrée future (à confirmer avec Samuel).
- Aucune authentification : l'application ne doit pas être exposée hors du réseau interne.
- Base en mémoire : les données sont perdues à chaque redémarrage.
```

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 et le contrat documenté sont ce qui fait passer Hamac de l'une à l'autre.

Erreurs fréquentes

- **Renommer** `workingDays` **dans l'API** pour l'aligner sur la page : vous cassez le contrat publié et les tests.
- **Oublier de tester** `response.ok` : `fetch` ne lève pas d'erreur pour un `409`, et la page affiche une ligne vide au lieu du message.

- **Oublier l'en-tête** `Content-Type` **dans** `createRequest` : l'API répond `415`, et la demande n'est jamais créée.
- **Ajouter** `@CrossOrigin("*")` : la page n'en a pas besoin, et n'importe quel site pourrait appeler l'API depuis le navigateur de vos utilisateurs.
- **Afficher un message avec** `innerHTML` **ou** `th:utext` : rien ne casse aujourd'hui, mais la porte est ouverte aux injections de script.
- **Oublier** `Math.max` **dans** `InternshipPolicy` : le solde de Zoé vaut -2 jours, et le test du stage qui n'a pas commencé échoue.
- **Modifier** `LeaveService` **ou le résolveur pour gérer les stages** : le scan des composants et la liste des politiques rendent ces modifications inutiles.