Compte-rendu des améliorations apportées au visualiseur Neo4j / Neovis.js

1. Sécurisation de la génération d'`index.html`

Dans la branche originale, la méthode `buildVizHtmlFile` lisait le gabarit `index_pattern.html` à l'aide de : `getResource("/index_pattern.html").getFile()`

Puis ouvrait le fichier via de simples FileReader / FileWriter. Cette technique échoue dès qu'un caractère doit être encodé dans le chemin (espace → %20, accent, etc.), car la chaîne retournée n'est pas un véritable chemin de fichier et l'appel se solde par un **FileNotFoundException**.

Nous avons :

- converti l'URL de ressource en **URI** (getResource(...).toURI()),
- créé un **Path** avec Path.of(uri) — ce qui décode automatiquement les séquences %xx,
- remplacé FileReader/FileWriter par **Files.newBufferedReader / Files.newBufferedWriter** dans un bloc *try-with-resources*.

Ainsi, la lecture/écriture du gabarit fonctionne quelles que soient la plate-forme et la présence de caractères spéciaux ; les flux sont toujours fermés proprement, éliminant les fuites de ressources.

2. Robustesse accrue des requêtes Cypher générées (Solver.Issue)

Dans la version Pauline :

- le motif d'arête était figé : -[:used]-> (orientation imposée) ;
- les identifiants étaient interpolés sans guillemets — risque de syntax error si un nom contient un espace ou un symbole spécial ;
- la chaîne finale gardait retours à la ligne / tabulations, rendant le Cypher illisible ou fragile côté JavaScript.

Nos corrections :

1. **Motif non orienté** -[u:used]- : la requête trouve l'arête quel que soit le sens stocké.
2. **Helper quoted()** : entoure systématiquement les valeurs de '...', supprimant tout souci d'échappement.
3. **Compactage automatique** : replaceAll("\\s+", " ").trim() produit une requête sur une seule ligne, propre pour l'injection dans Neovis.js.

Le résultat est plus résilient face aux variations de schéma et maintenable sur le long terme.

3. Visualisation : labels compacts / infobulles exhaustives

3.1 Limites de l'approche initiale

- Sur le graphe : seul **`name`** apparaît sur les nœuds, seul **`ctx`** sur les arêtes.
- L'infobulle (`title`) affichait **une propriété figée** (`TG`, `TC`, `TU` ou `T`) définie à la main pour chaque type de relation.
- Toute nouvelle propriété nécessitait de modifier le code JavaScript.

3.2 Mécanisme introduit

Nous avons injecté, via l'API avancée de NeoVis, une fonction générique :

```
const fullDetails = item =>
  Object.entries(item.properties)
        .map(([k,v]) => `${k}: ${v}`)
        .join('\n');
```

Cette fonction est branchée sur **title** pour *tous* les labels et *toutes* les relations :

```
l[NeoVis.NEOVIS_ADVANCED_CONFIG] = { function: { title: fullDetails } };
```

Le label visuel reste minimal (lisible) mais, au survol, la tooltip déroule **l'intégralité** des métadonnées présentes dans Neo4j, y compris TB, TE ou tout champ ajouté à l'avenir — sans qu'une seule ligne de JavaScript ne doive être modifiée.

3.3 Illustration