🔧 Ajuste Obrigatório – Tornar o Ranking confiável (substituir `rankBetween` atual)

Problema identificado

O módulo atual `packages/shared/rank.ts` **não é confiável** para ordenação no Kanban.

Motivos:

- `rankInitial()` e `rankBetween()` usam escalas/bounds inconsistentes (`h00000` vs `000000/zzzzzz`)
- o algoritmo atual tende a falhar em casos reais:
  - muitas inserções "entre" os mesmos dois itens
  - ranks crescendo indefinidamente
  - risco de colisão / rank inválido
  - comportamento imprevisível com string compare

Isso pode quebrar reorder e gerar bugs difíceis de rastrear.

Decisão técnica (não negociável)

Vamos substituir o ranking por um sistema **numérico com gaps + reindex local**, simples e robusto.

Regras do novo ranking

- `rank` passa a ser **INTEGER** (no banco e nos DTOs).
- Em cada lista, ranks crescem com "gap" fixo: **1000**.
  - Ex.: 1000, 2000, 3000…
- Ao mover/inserir:
  - se houver espaço entre `beforeRank` e `afterRank`, usar o meio.
  - se não houver espaço (diferença <= 1), executar **reindex daquela lista** e tentar novamente.

Esse modelo é extremamente estável e fácil de manter.

Mudanças necessárias (passo a passo)

1) Banco de dados (Prisma)

- Alterar `rank` de `String` para `Int` em `List` e `Card`.
- Criar migração.
- Garantir índice: `(listId, rank)` em `Card` e `(boardId, rank)` em `List`.

Observação: manter a ordenação sempre por `rank asc`.

---

2) Contratos em `packages/shared`

- Atualizar schemas/DTOs para `rank: z.number().int()`.
- Remover `rank.ts` antigo e criar um novo módulo:

`packages/shared/rank.ts`:

```
export const RANK_GAP = 1000; export function rankInitial(): number { return RANK_GAP;
} export function rankBetween(before: number | null, after: number | null): number |
null { // Retorna null quando não há espaço e é necessário reindex }
```

Implementação:

- Se `before == null && after == null` => `rankInitial()`
- Se `before == null` => `Math.floor(after / 2)`
- Se `after == null` => `before + RANK_GAP`
- Se `after - before > 1` => `before + Math.floor((after - before) / 2)`
- Se `after - before <= 1` => retornar `null` (sem espaço → precisa reindex)

---

3) Reindex local (obrigatório)

Implementar no backend (service/repository):

ReindexListCards(listId):

- Buscar cards ativos da lista ordenados por `rank asc`.
- Reatribuir ranks: `1000, 2000, 3000...` na ordem atual.
- Persistir em transação.

O mesmo vale para **reorder de listas** dentro de um board (se aplicável).

---

4) Atualizar endpoints de move/reorder

Quando mover/inserir card/list:

- Calcular `beforeRank` e `afterRank` a partir dos vizinhos (IDs `beforeCardId/afterCardId`).
- Chamar `rankBetween(beforeRank, afterRank)`.

Se retornar `null`:

1. executar `reindexListCards(listId)`
2. recarregar os ranks
3. recalcular e aplicar o `rankBetween` novamente
4. persistir

Tudo dentro de uma transação.

Testes obrigatórios (unit + integração)

Unit tests (shared)

Criar testes para `rankBetween`:

- inserir no começo (`before=null, after=1000` => 500)
- inserir no fim (`before=3000, after=null` => 4000)
- inserir no meio (`before=1000, after=2000` => 1500)
- caso sem espaço (`before=1000, after=1001` => null)

Integração (api)

Criar um teste que force reindex:

- gerar lista com ranks 1000 e 1001
- tentar inserir entre eles
- backend deve reindexar e inserir com rank válido

Critérios de Aceitação

- ✅ Ordenação estável por `rank asc`
- ✅ Inserções repetidas entre os mesmos itens não quebram o sistema
- ✅ Quando faltar espaço, reindex local acontece automaticamente
- ✅ Nenhum rank é string
- ✅ Sem crescimento infinito de rank
- ✅ Testes passam

Nota importante

Se existir dado antigo com rank string, criar um script de migração:

- ordenar pelo rank antigo (ou `createdAt` se necessário)
- atribuir ranks 1000,2000,3000.