Desafio Técnico Backend

Projeto Kanban

Visão geral

Você irá construir uma API back-end (Java) para gerenciar **Projetos** através de um **quadro Kanban** e **indicadores e Resumo (Opcional)**. O sistema precisa refletir regras de negócio específicas, suportar transições de status conforme tabela abaixo, calcular métricas (ex.: percentual de tempo restante) e expor documentação (Swagger/OpenAPI). O projeto deve ser **dockerizado**, coberto por **testes automáticos** e com **histórico de commits** claro.

Stack mínima: Java 8 (ou superior), Spring Boot (ou outro framework), Maven/Ant, JUnit, Git, Docker.
Integrações: APIs REST e GraphQL (diferencial).
Entrega: repositório público GitHub, README, imagens/coleções (Swagger, Postman/Insomnia, Diagramas da solução proposta).

Escopo funcional

Entidades do modelo de negócio:

Projeto

- Id (UUID ou Long)
- Nome
- Status: A iniciar, Em Andamento, Atrasado, Concluído
- Responsável (pode ter um ou mais responsáveis)
- Datas:
    - Início Previsto
    - Térmico Previsto
    - Início Realizado
    - Térmico Realizado
- Dias de Atraso
- Percentual de tempo restante

Responsável

- Id (UUID ou Long)
- Nome
- Email
- Cargo
- Secretaria (CRUD de secretaria é um diferencial)

Regras de negócio

Definição de cada status de projeto:

- A iniciar: as datas de início realizado e térmico realizado são vazias;
- Em andamento:
  - Início realizado preenchido, e
  - Térmico previsto é maior que a data de hoje, e
  - Térmico realizado é vazio.
- Atrasado:
  - Início previsto é menor que a data de hoje e início realizado é vazio, ou,
  - Término previsto é menor que a data de hoje e término realizado é vazio.
- Concluído:
  - Térmico realizado preenchido

Obs: Ao editar as datas de início previsto, término previsto, início realizado e térmico realizado, o status deve ser automaticamente atualizado seguindo as regras de transição que estão definidas nas seções abaixo.

Tabela de transição de status (ações e validações)

De → Para	Ações Automáticas	Validações / Erros
A iniciar → Em andamento	Definir Início Realizado = data de hoje.	-
A iniciar → Atrasado	-	**Erro** se hoje < Início Previsto (não pode marcar Atrasado antes do início previsto).
A iniciar → Concluído	Definir Término Realizado = hoje.	-
Em andamento → A iniciar	Início Realizado = null.	-

Em andamento → Atrasado	-	**Erro** solicitando: ou remover Início Realizado (volta a 'não iniciado' com **atraso se cabível**) **ou** ajustar Início/Término Previsto para data **< hoje**.
Em andamento → Concluído	Definir Término Realizado = hoje.	-
Atrasado → A iniciar	-	**Erro** solicitando: remover Início Realizado **e** ajustar Início/Término Previsto para data **> hoje**.
Atrasado → Em andamento	-	**Erro** solicitando: ajustar Início/Término Previsto para data **> hoje**.
Atrasado → Concluído	Definir Término Realizado = hoje.	-
Concluído → A iniciar	-	**Erro** solicitando: remover Término Realizado **e** ajustar Início/Término Previsto para data **> hoje**.
Concluído → Em andamento	Término Realizado = null.	**Validar** se, ao remover, o projeto **não** fica Atrasado; se ficar, bloquear e solicitar ajuste de datas.
Concluído → Atrasado	Término Realizado = null.	**Só permitir** se, ao remover, as regras classificarem como Atrasado, se não bloquear e solicitar ajuste de datas.

Nota: Em toda transição, **recalcular** status por regras após aplicar efeitos; se o status final não corresponder ao solicitado (por inconsistência de datas), **bloquear** e retornar mensagem clara com dicas do que ajustar.

Cálculo - Percentual de tempo restante

Fórmula:

Se Início Previsto e Término Previsto preenchidos:

total = diasEntre(Início Previsto, Término Previsto)
usado = diasEntre(Início Previsto, hoje)
restante = total - usado

- Se total <= 0: retornar 0 (evitar divisão por zero).
- Em A iniciar (sem datas) retornar 0.
- Em Concluído pode retornar 0 (não resta tempo).
- Se hoje > término previsto e não Concluído, pode ficar 0 (tempo excedido).

Cálculo - Dias de atraso

- Se Término Realizado nulo e Término Previsto < hoje: diasAtraso = diasEntre(Término Previsto, hoje).
- Se Concluído: diasAtraso = 0
- Em A iniciar (sem datas) = 0.

Entregáveis

- Se não conseguir entregar tudo o que foi pedido, não deixe de entregar o que for capaz de realizar.

Etapa 1 - CRUD de Projeto e Responsável

- Endpoints REST
- Regras:
  - Ao criar/editar projeto: recalcular status, diasAtraso, percentualTempoRestante.
  - Validar e-mail do responsável (e-mail único).
  - Auditoria mínima: createdAt, updatedAt.
- Swagger/OpenAPI publicado em /swagger-ui ou /api-docs
- Testes: JUnit para services e controllers; mocks; cobertura mínima 70%

Etapa 2 - Kanban (listar por status + transições)

- Endpoints REST
- Regras:
  - Transição de status para os projetos;
  - Obter listagem dos projetos por status do kanban;
  - Aplicar a Tabela de Transição acima (com mensagens de erro claras).
  - Movimentação por drag-and-drop (se houver UI - diferencial).
- Testes cobrindo: casos felizes, bloqueios (erros), confirmações obrigatórias e cálculo de status/métricas.

Etapa 3 - Indicadores e Resumos (Opcional)

- Endpoints REST
- Operações no GraphQL (diferencial);
- Regras:
  - Média de dias de atraso por status;
  - Quantidade de Projetos por status;

    - Formule novos endpoints e operações, pois será um diferencial.
- Testes cobrindo casos de sucesso e erros.

Requisitos não-funcionais

- Qualidade de código: Clean Code, SOLID, Design Patterns.
- Exceções: tratamento e padronização de erros dos serviços.
- Segurança: validação de inputs e logs adequados.
- Performance: paginação em listagens; índices em colunas de filtro.
- Observabilidade (diferencial): Actuator (se Spring Boot), métricas em Prometheus/Grafana.

Testes automatizados

- Camadas:
    - Unidade (cálculo de status, percentuais, atrasos).
    - Integração (repositories/DAO, transações).
    - API (WebMvcTest ou Testcontainers para DB real, são diferenciais).

Documentação e README

- README (obrigatório):
    - Visão geral do problema e decisões técnicas.
    - Como rodar (Docker), como testar e como acessar Swagger.
    - Estrutura de pastas e convenções.
    - Limitações e próximos passos.
- Swagger/OpenAPI com exemplos de requests/responses, schemas, e mensagens de erro.
- Histórico de commits: granular (feature flags, refactors, testes, docs).

Diferenciais

- UI simples (React/React Native/Angular) consumindo a API (Kanban drag-and-drop).
- CRUD de Secretaria como entidade própria.
- Filtros avançados (por secretaria, período, responsável, texto).
- GraphQL (schema para Projetos/Responsáveis).
- Observabilidade (Prometheus/Grafana), Logs estruturados.
- CI/CD (GitHub Actions com build, testes e análise).
- Simulação de PR com code review.
- Processos BDD e TDD.
- Serviço de autenticação do responsável.