Ideia de Projeto:

Um Simulador de Carreira Militar concebido para apoiar o planeamento e a gestão dos efetivos por quadro especial, permitindo simular a progressão na carreira dos militares, visualizando e antevendo a evolução ao longo do tempo.

A simulação poderá ser feita para um só militar ou para um quadro especial específico. No caso da simulação apenas para um militar, é gerado um relatório com a data previsível das suas promoções, até ao momento da sua passagem à reserva. Na simulação de um Quadro Especial, é definido o período a simular, e no final é gerado um relatório acerca do número de passagens à reserva, número de promoções e o número de militares por posto ao fim da simulação.

Para realizar a simulação o sistema possibilita o registo dos dados individuais dos militares como a identificação, o quadro especial a que pertence, a idade, o número de anos no posto e o histórico da sua carreira militar. Permite ainda definir parâmetros para a simulação, nomeadamente o número de lugares disponíveis por posto dentro de um determinado quadro especial, e condições estatutárias por posto para a passagem à reserva ou promoção, tais como os limites de idade e os tempos máximos e mínimos de permanência em cada posto.

> O major Azevedo abriu o Simulador de Carreira Militar. Em segundos, o sistema mostrou o tempo em cada posto, a previsão da próxima promoção e a data da sua passagem à reserva.

Entidades/ações candidatas a classes/comportamentos

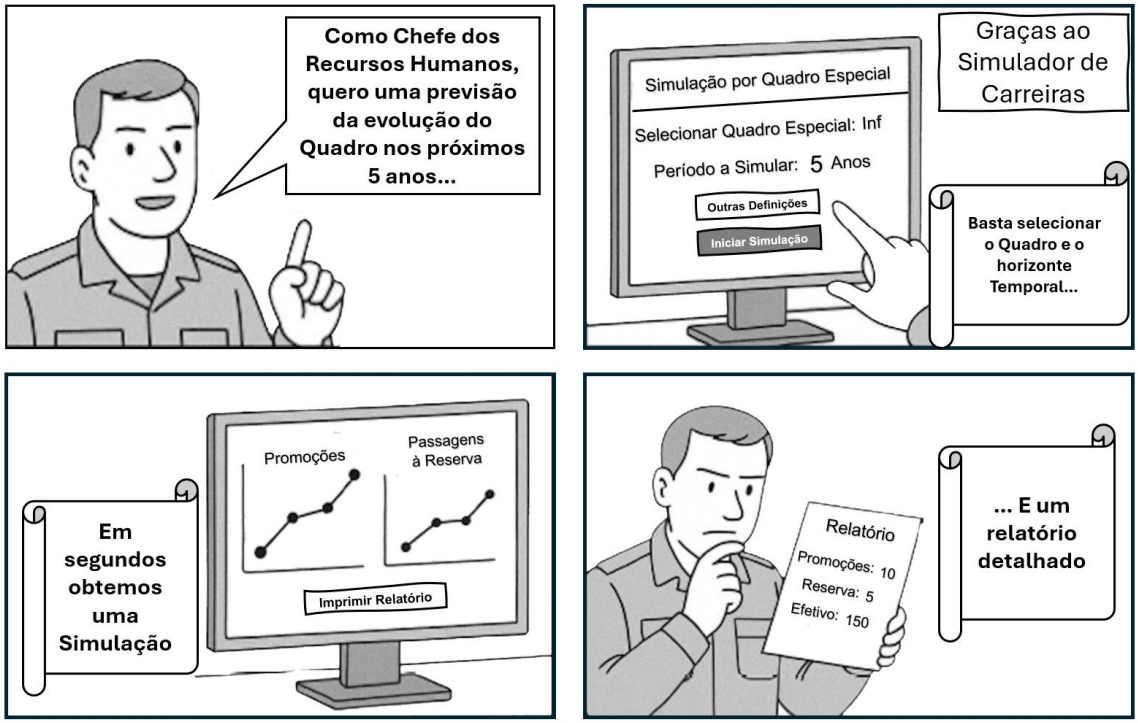
Militar – É **criado** e **atualizado** ao longo da simulação; reúne dados dos militares, identificação, Quadro Especial, posto atual e histórico de carreira de cada militar.
Posto – É **consultado** (com parâmetros de tempo e idade) e **utilizado** para verificar condições de promoção ou de passagem à reserva.
Quadro Especial – É **consultado** e **selecionado**; define o conjunto de vagas disponíveis em cada posto.
Gestor de Quadros – **Consulta** Quadro Especial e Militares e c**alcula** o número de vagas por posto e quadro.
Gestor de Carreiras – **Verifica** condições de promoção e **executa** promoções e passagens à reserva, dos militares.
Simulador de Carreira – **Executa** a progressão anual; **atualiza** idade, tempo de posto e serviço; **gera** resultados parciais.
Controlador de Simulação – **Coordena** a execução global da simulação e **produz** relatórios e estatísticas finais.
Repositório de Dados – **Guarda** e **carrega** informações de militares e vagas; permite a **persistência** dos resultados e **parametrizar** simulações.

User story 1 – Carreira de um militar

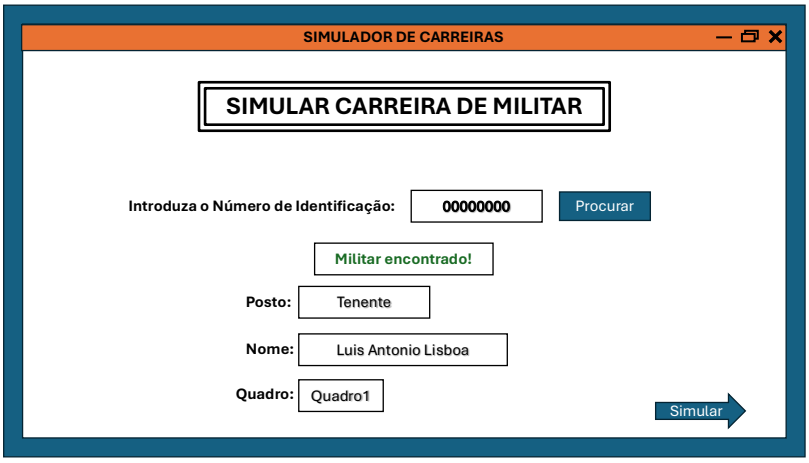
Como elementos de um Estado-Maior na área dos recursos humanos, queremos saber qual a previsão de um militar em subir de posto, saber quando passará à reserva, o tempo que passou nos postos passados e que irá passar em postos futuros.

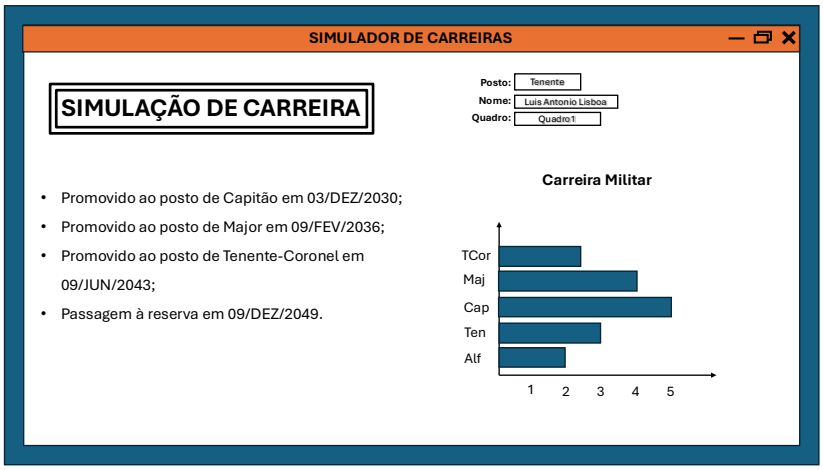
User story 2 – Simular a progressão na carreira

Como elementos de um Estado-Maior na área dos recursos humanos, queremos saber a evolução anual do quadro, quantos militares são promovidos, os que atingem a idade limite e as vagas criadas por posto, e antecipar desequilíbrios na estrutura do quadro.

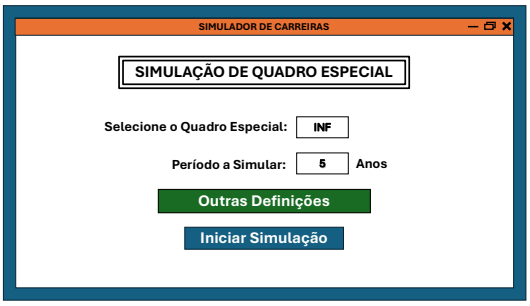


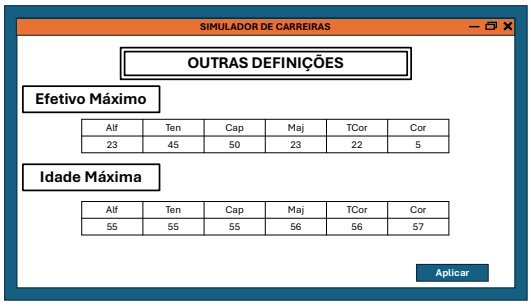
Mockups da user story 1

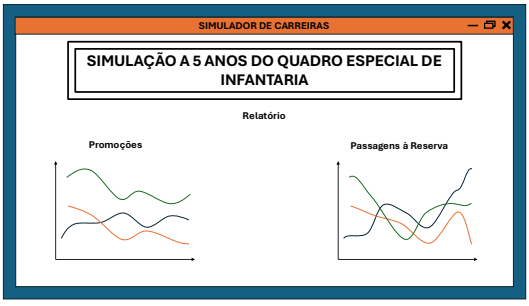




Mockups da user story 2







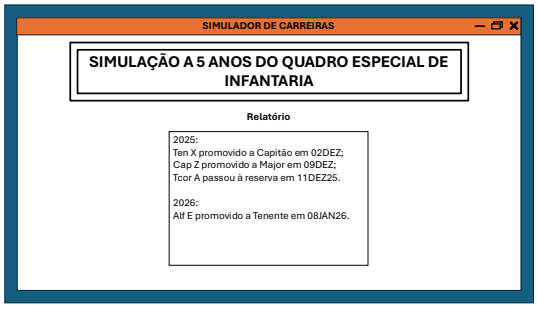
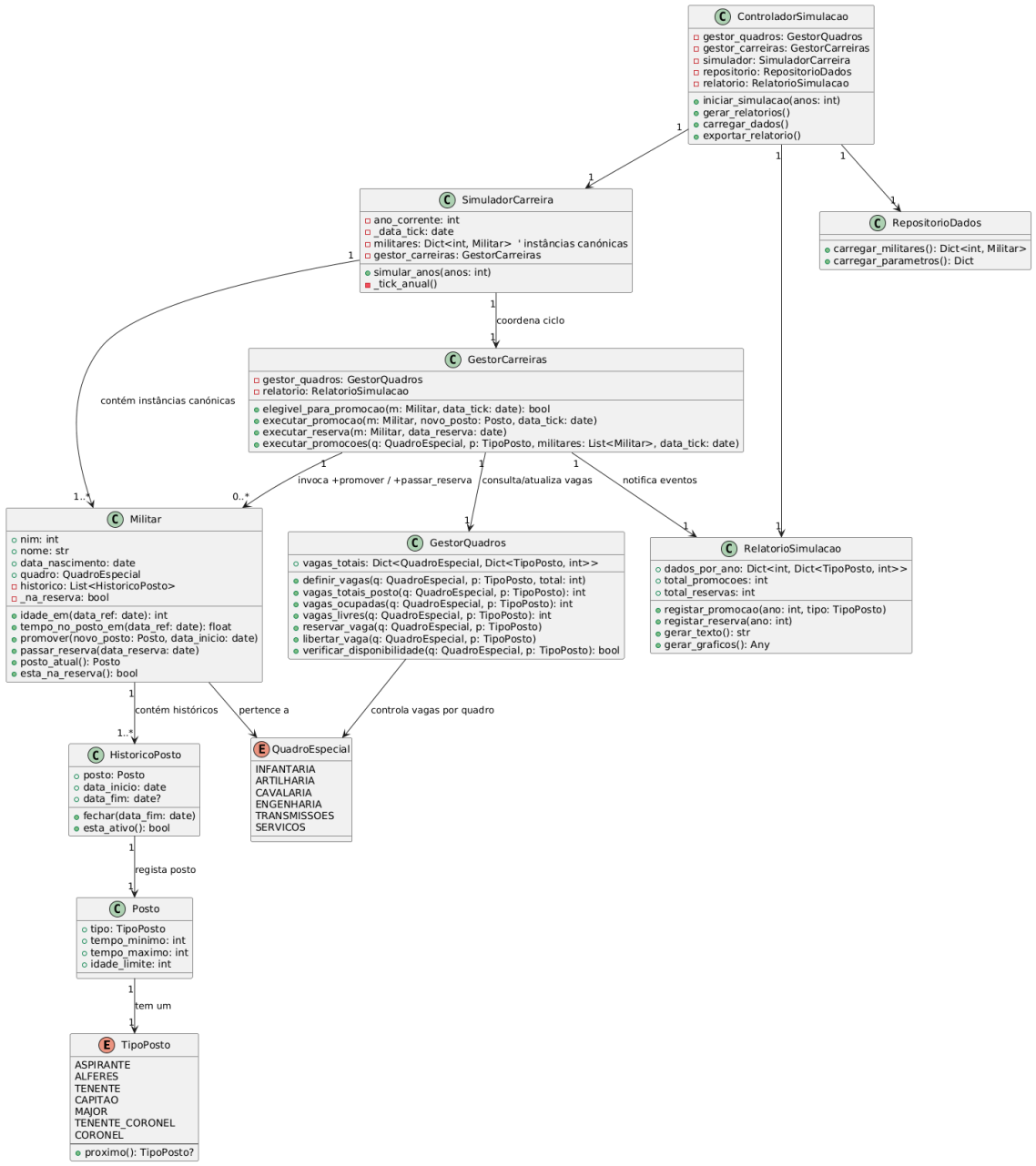


Diagrama de classes



Protótipos em Python

```
 1. from datetime import date
 2.
 3. # ======================
 4. # CAMADA DE DOMÍNIO
 5. # ======================
 6. class Posto:
 7.     def __init__(self, tipo, tempo_minimo, idade_limite, proximo=None):
 8.         self.tipo, self.tempo_minimo, self.idade_limite, self.proximo = tipo,
tempo_minimo, idade_limite, proximo
 9.
10. class HistoricoPosto:
11.     def __init__(self, posto, data_inicio):
12.         self.posto, self.data_inicio, self.data_fim = posto, data_inicio, None
13.     def fechar(self, data_fim): self.data_fim = data_fim
14.     def esta_ativo(self): return self.data_fim is None
15.
16. class Militar:
17.     def __init__(self, nim, nome, data_nascimento, quadro, posto_inicial,
data_inicio):
18.         self.nim, self.nome, self.data_nascimento, self.quadro = nim, nome,
data_nascimento, quadro
19.         self._historico, self._na_reserva = [HistoricoPosto(posto_inicial,
data_inicio)], False
20.     def posto_atual(self): return self._historico[-1].posto
21.     def idade_em(self, data_ref): return data_ref.year - self.data_nascimento.year
22.     def promover(self, novo_posto, data_inicio):
23.         self._historico[-1].fechar(data_inicio)
24.         self._historico.append(HistoricoPosto(novo_posto, data_inicio))
25.     def passar_reserva(self, data_reserva):
26.         self._historico[-1].fechar(data_reserva)
27.         self._na_reserva = True
28.
29. # ======================
30. # CAMADA DE GESTÃO
31. # ======================
32. class GestorCarreiras:
33.     def __init__(self, ano_corrente): self.ano_corrente = ano_corrente
34.     def elegivel_para_promocao(self, m):
35.         anos = self.ano_corrente - m._historico[-1].data_inicio.year
36.         return anos >= m.posto_atual().tempo_minimo
37.     def executar_promocoes(self, militares):
38.         data_tick = date(self.ano_corrente, 1, 1)
39.         for m in militares:
40.             if m._na_reserva: continue
41.             posto = m.posto_atual()
42.             if m.idade_em(data_tick) >= posto.idade_limite:
43.                 m.passar_reserva(data_tick)
44.                 print(f"{m.nome} passou à reserva ({posto.tipo})")
45.             elif self.elegivel_para_promocao(m) and posto.proximo:
46.                 m.promover(posto.proximo, data_tick)
47.                 print(f"{m.nome} promovido para {posto.proximo.tipo}")
48.             else:
49.                 print(f"{m.nome} mantém-se em {posto.tipo}")
50.
51. # ======================
52. # INSTÂNCIAS
53. # ======================
54. alferes = Posto("Alferes", 2, 52)
55. tenente = Posto("Tenente", 3, 54)
56. alferes.proximo = tenente
57. joao = Militar(1001, "João", date(1975,1,1), "Infantaria", alferes, date(2023,1,1))
58. maria = Militar(1002, "Maria", date(1985,1,1), "Transmissões", alferes,
date(2024,1,1))
59. GestorCarreiras(2025).executar_promocoes([joao, maria])
60.
```

Mapa de decisões

Incoerência	Consequência	Limite Aceite	Princípio OO
Mutação do estado de carreira espalhada: Militar.promover, GestorCarreiras, GestorQuadros	O estado de carreira (promoção) podia ser alterado em vários pontos, tornando difícil garantir invariantes de consistência (ex.: vaga atualizada mas histórico não).	Foi centralizada num **ponto transacional único** (GestorCarreiras.executar_promocoes()), que orquestra todas as alterações de estado e assegura atomicidade.	**Encapsulamento / Responsabilidade Única**
Instâncias canónicas e identidade por NIM entre SimuladorCarreira e RepositorioDados	Risco de duplicação de objetos do mesmo militar (NIM) quando carregados ou simulados, comprometendo a unicidade da identidade.	O SimuladorCarreira mantém a **coleção canónica de militares**, e o RepositorioDados apenas carrega dados deduplicados, garantindo **entrada única e unificação por NIM**.	**Identidade / Encapsulamento**
Dois indicadores de estado ativo/reserva (_na_reserva e HistoricoPosto.esta_ativo())	Existência de duas fontes de verdade; possível divergência entre o flag e o histórico.	_na_reserva passou a ser **derivado do histórico ativo**, eliminando mutações diretas e garantindo consistência temporal.	**Estado / Encapsulamento**
Notificação de eventos (promoção/reserva) ao RelatorioSimulacao	Se os eventos fossem registados antes do commit, o relatório podia refletir ações falhadas ou incoerentes.	O relatório é agora **notificado apenas após a confirmação da transição**, armazenando valores imutáveis (contadores e DTOs).	**Encapsulamento / Estado**
TipoPosto.proximo() define progressão enquanto GestorCarreiras decide promoções	Duplicação de responsabilidades (ordem estática vs. política dinâmica), podendo gerar divergências subtis.	Mantém-se a **delimitação clara**: proximo() define apenas a **ordem hierárquica estática**, e o GestorCarreiras gere **as condições de elegibilidade**.	**Responsabilidade Única / Abstração**
Ambiguidade classe↔instância (Posto "tem um" TipoPosto; Militar "pertence a" QuadroEspecial)	Confusão entre níveis de modelação (metaclasse vs. associação simples); risco de interpretação errada no diagrama.	Relações são **unidirecionais e descritivas** ("tem um" / "pertence a"); TipoPosto e QuadroEspecial mantêm-se como **enums fixos**, apenas referenciais.	**Classe vs. Instância / Abstração**

Síntese reflexiva

Incoerência: Promoção e passagem à reserva atravessam múltiplos agregados sem uma fronteira transacional explícita. No modelo atual, a execução de uma promoção (GestorCarreiras.executar_promocao(...)) envolve três entidades independentes: o Militar, que altera o seu histórico e estado; o GestorQuadros, que atualiza a ocupação das vagas; e o RelatorioSimulacao, que regista o evento. No entanto, estas operações não estão **unificadas sob uma mesma transação lógica** — o diagrama define as relações e responsabilidades, mas não estabelece a *ordem garantida de execução* nem a *regra de atomicidade* ("tudo-ou-nada") entre estas mutações de estado. Isto faz com que o sistema dependa implicitamente da boa ordem de chamadas e da ausência de falhas durante a execução, o que é uma violação indireta do princípio de **Encapsulamento** comportamental: o domínio não garante a sua própria coerência, mas delega-a a quem o chama.

Consequência: Ausência de uma fronteira transacional formal introduz o risco de **inconsistência parcial entre agregados**. Em cenários de erro, o estado do sistema pode ficar dividido: por exemplo, o GestorQuadros marca uma vaga como ocupada, mas o Militar não atualiza o seu HistoricoPosto, criando uma vaga "fantasma"; ou o RelatorioSimulacao regista uma promoção que nunca foi efetivada no estado real. Estes desfasamentos tornam os invariantes estruturais do modelo vulneráveis — como "máx. 1 histórico ativo por militar", "número de vagas ocupadas = número de militares em posto ativo", ou "todas as promoções registadas têm correspondência no histórico". A médio prazo, isso gera efeitos acumulativos de corrupção semântica: relatórios inconsistentes, simulações não reprodutíveis e dificuldade de testar regras de progressão. Além disso, a dispersão da responsabilidade (cada classe a alterar uma parte do estado) aumenta o acoplamento implícito e reduz a testabilidade e previsibilidade do sistema, comprometendo o encapsulamento e a rastreabilidade de erros.

Limite aceite: Esta fragilidade pode ser mitigada através da criação de uma fronteira transacional explícita dentro do GestorCarreiras, que passa a ser o único orquestrador autorizado a coordenar as mutações de estado entre Militar, GestorQuadros e RelatorioSimulacao. A mitigação recomenda que o GestorCarreiras implemente uma operação agregada (ex.: executar_promocao(m: Militar, novo_posto: Posto, data_tick: date)) que centralize, por ordem: (1) validação de elegibilidade; (2) libertação e ocupação de vagas; (3) invocação de Militar.promover(...); (4) registo de evento no relatório apenas após sucesso das etapas anteriores. Todas estas operações devem ser executadas dentro de um contexto de commit único, com rollback lógico compensatório em caso de falha (ex.: repor vaga e desfazer histórico). Esta abordagem reforça a **Responsabilidade Única** do GestorCarreiras, respeita o **Encapsulamento** de cada agregado (que só é manipulado através das suas APIs públicas) e mantém a Identidade dos objetos coerente com os seus invariantes de integridade.

Evidências incluídas no documento: estou a incluir *storyboards* e *mockups*, excertos de código (menos de 60 linhas).

Princípios utilizados explicitamente nesta reflexão, acima: *encapsulamento*; *identidade*, *responsabilidade única*. Identifiquei-os a negrito.