***Relatório Técnico Etapa 1 do trabalho de Sistemas distribuídos***

#### **1. Arquitetura do Sistema**

O sistema foi desenvolvido utilizando a arquitetura MVC distribuída:

* Model: BankAccount.java e Database.java – responsáveis por armazenar e manipular os dados das contas bancárias.
* View: Menu.java – interface de linha de comando (CLI) para interação com o usuário.
* Controller: Group.java – integra a lógica de replicação e sincronização usando JGroups.

O sistema é distribuído por meio do middleware JGroups, com replicação horizontal: múltiplas instâncias do servidor participam do mesmo canal de comunicação e compartilham o estado do sistema (lista de contas, saldo, extrato, etc.).

#### **2. Principais Decisões de Projeto**

* JGroups foi adotado como mecanismo de replicação e comunicação entre nós, garantindo consistência e tolerância a falhas.
* Toda a lógica de persistência foi encapsulada na classe Database.java, que salva e carrega os dados em um arquivo binário (banco\_estado.dat), permitindo retomar o estado do sistema em caso de desligamento.
* O sistema identifica cada cliente por CPF, garantindo unicidade e impedindo duplicação de contas.
* A comunicação entre nós utiliza os métodos getState() e setState() do JGroups, permitindo sincronização automática do estado ao ingressar em um canal.

#### **3. Pontos Fortes da Solução**

* Replicação de Estado: novas instâncias sincronizam automaticamente o estado via JGroups.
* Simples e funcional: a interface em terminal permite testes rápidos das operações bancárias.
* Persistência local confiável: garante recuperação do sistema após falhas.
* **Evita inconsistência de saldo**: usa sincronização explícita com synchronized para evitar condições de corrida nas transferências.

#### **4. Pontos Fracos da Solução**

* Comunicação RPC ainda não implementada: cliente e servidor compartilham o mesmo processo.
* Canal único: há apenas um canal JGroups; não há segmentação por camadas como "Controle" e "Dados".
* Sem mecanismos de segurança (criptografia, autenticação reforçada).
* Falta de testes de carga (MPerf) e análise de desempenho.
* Ausência de balanceamento de carga: o coordenador responde diretamente às requisições.

#### **5. Justificativa da Pilha de Protocolos (JGroups)**

Foi utilizada uma pilha padrão simplificada com suporte a:

* UDP (Multicast) para descoberta e comunicação entre instâncias.
* GMS (Group Membership Service) para detecção de falhas.
* NAKACK2 para retransmissão e ordenação confiável de mensagens.
* STATE\_TRANSFER para replicação do estado global entre os nós.

A pilha foi configurada via config.xml, possibilitando fácil substituição ou ajuste dos protocolos conforme necessário na Etapa 2.

#### **6. Estado do Sistema Compartilhado**

O sistema compartilha entre os nós:

* O mapa completo de contas (ConcurrentHashMap com CPF → BankAccount)
* O histórico de operações de cada conta
* O montante total do banco (derivado do somatório dos saldos)

A sincronização ocorre automaticamente com getState()/setState() quando um novo nó entra no cluster JGroups.

### **📎 Anexos**

* Código-fonte completo (.java)
* Bytecodes (.class)
* config.xml
* Script de execução (run.sh)
* Arquivo de estado: banco\_estado.dat
* Biblioteca: jgroups-3.6.4.Final.jar