Documentação da Etapa 2

1. Estado Atual do Projeto:

Nessa etapa criei duas entidades principais como **processos independentes** e na **comunicação básica via sockets TCP**.

Atualmente, o projeto consiste em dois componentes principais funcionando:

- Um **Servidor de Livros (processo servidor)**: Atua como a entidade central que "detém" os dados dos livros. Ele está configurado para escutar requisições em uma porta específica (12345) e responder com informações sobre livros. É multi-threaded, capaz de processar requisições de vários clientes.
- Um **Cliente de Livros (processo cliente)**: Simula um consumidor das informações. Ele é capaz de se conectar ao servidor, enviar requisições para obter dados de livros e exibir as respostas recebidas. O cliente estabelece uma nova conexão com o servidor para cada comando enviado.

Ambos os componentes se comunicam utilizando **sockets TCP**, demonstrando a troca de mensagens confiáveis. O servidor consegue processar requisições para retornar um JSON simulado de livros, e o cliente pode iniciar e encerrar a comunicação de forma controlada.

2. Justificativa para a Escolha entre Processos e Threads:

Para essas entidades principais do meu projeto de site de livros (similar ao Letterboxd), optei por implementá-las como **processos independentes (instâncias separadas da JVM)**. Esta decisão foi pelos seguintes motivos:

- **Isolamento e Tolerância a Falhas:** Nas próximas etapas, vou implementar diferentes funcionalidades (como autenticação, busca de livros, gerenciamento de resenhas) em módulos. Ao separá-las em processos distintos, garanto que uma eventual falha ou erro em um módulo (por exemplo, no serviço de resenhas) não comprometa a disponibilidade ou estabilidade de outras partes críticas do sistema, como o serviço de busca de livros ou o login de usuários. Cada processo vai operar em seu próprio espaço de memória, o que minimiza significativamente o risco de corrupção de dados entre os componentes.
- **Escalabilidade Horizontal:** A arquitetura baseada em processos vai facilitar imensamente a distribuição do sistema em múltiplas máquinas físicas ou virtuais. Por exemplo, em futuras etapas, o serviço de busca de livros, que pode

enfrentar alta demanda, poderá ser alocado em um servidor dedicado, enquanto o serviço de autenticação reside em outro. Isso nos permite escalar cada componente de forma independente, otimizando o uso de recursos conforme a necessidade de cada funcionalidade.

- **Manutenção e Implantação Modular:** A separação em processos distintos permite que cada módulo seja desenvolvido, testado, implantado e atualizado de forma independente. Isso acelera o ciclo de desenvolvimento, simplifica a manutenção e reduz o risco de regressões ao introduzir novas funcionalidades ou corrigir bugs.

Dentro do processo servidor, empreguei **threads** para gerenciar concorrência interna (atendimento à múltiplas requisições simultâneas no servidor), combinando o isolamento dos processos com a eficiência das threads para tarefas paralelas.

3. Descrição da Comunicação Via Sockets:

A comunicação nesta etapa é estabelecida entre o **Cliente de Livros** e o **Servidor de Livros**.

- **Componentes Envolvidos:**
    - **Servidor de Livros:** Atua como o lado "servidor" da conexão, escutando em uma porta específica e gerenciando múltiplas conexões a partir das threads.
    - **Cliente de Livros:** Atua como o lado "cliente", iniciando uma nova conexão com o servidor para cada requisição.
- **Tipo de Socket Utilizado: TCP (Transmission Control Protocol)**
    - **Justificativa para TCP:** A escolha do TCP foi crucial para a comunicação neste projeto, pois os dados trocados (requisições de livros, JSON de resposta) exigem **confiabilidade e garantia de entrega**. O TCP oferece:
        - **Entrega Confiável:** Garante que todos os pacotes de dados enviados cheguem ao destino e na ordem correta, retransmitindo pacotes perdidos ou fora de ordem.
        - **Controle de Fluxo:** Evita que um emissor rápido sobrecarregue um receptor lento.
        - **Controle de Congestionamento:** Ajuda a evitar o colapso da rede.
    - **Orientação à Conexão:** Embora o TCP seja orientado à conexão, no modelo atual, uma nova conexão é estabelecida para cada requisição do

cliente. A conexão é mantida apenas pelo tempo necessário para enviar a requisição e receber a resposta, sendo então fechada. Isso é ideal para interações de curta duração, onde a confiabilidade é prioritária.

- No futuro, se houver necessidade de funcionalidades que priorizem velocidade em detrimento de confiabilidade (ex: notificações muito rápidas que não precisam de garantia de entrega), o UDP poderá ser considerado para esses caso. No entanto, para a troca de dados estruturados como requisições e respostas de API (JSON), escolhi o TCP.

- **Estrutura das Mensagens:** As mensagens são atualmente estruturadas como **strings simples enviadas linha a linha** através dos streams de entrada e saída dos sockets.
  - **Requisições do Cliente para o Servidor:**
    - GET_LIVROS: Solicita ao servidor que retorne os dados dos livros disponíveis.
    - SAIR: Sinaliza ao cliente que ele deve encerrar sua própria execução (não é enviado ao servidor para fechar a conexão, pois a conexão é fechada após cada requisição).
    - Qualquer outra string é tratada como um comando desconhecido.
  - **Respostas do Servidor para o Cliente:**
    - Para GET_LIVROS: Uma string no formato **JSON** contendo um array de objetos Livro (ex: [{"titulo":"O Pequeno Príncipe","autor":"Antoine de Saint-Exupéry"},{"titulo":"1984","autor":"George Orwell"}]).
    - Para SAIR: Nenhuma resposta específica é enviada, pois o comando é tratado localmente pelo cliente para encerrar sua execução.
    - Para comandos desconhecidos: Uma string indicando "Comando desconhecido: [comando recebido]".

4. Instruções de Compilação e Execução:

Passos para compilar e executar o projeto:

1. **Abra o Terminal/Prompt de Comando:** Abrir o terminal no mesmo local da pasta src
2. **Compilar o Projeto:** Utilizar o comando javac para compilar os arquivos fonte:

- javac ServidorLivros.java ClienteLivros.java

Isso criará os arquivos .class correspondentes (ServidorLivros.class e ClienteLivros.class) no mesmo diretório.

3. **Executar o Servidor (Primeiro Processo):** Abrir **um novo terminal** (serão terminais separados para o servidor e para o cliente). No novo terminal, navegue até o diretório novamente e execute a classe ServidorLivros.
    - java ServidorLivros

Retorno: Servidor de Livros iniciado na porta 12345... e Esperando por conexão de cliente...

4. **Executar o Cliente (Segundo Processo):** No **outro novo terminal,** navegue até o diretório do projeto e execute a classe ClienteLivros.
    - java ClienteLivros

Retorno: Cliente de Livros iniciado... e Conectado ao servidor em localhost:12345.

5. **Interação:** No terminal onde o cliente está rodando, tem as opções de comandos:
    - GET_LIVROS: O cliente enviará a requisição ao servidor e exibirá o JSON de livros recebido.
    - SAIR: para que o cliente encerre a conexão e finalize sua execução.