Relatório Técnico da Aplicação de Gestão de Tarefas

1. **Identificação da Equipe**
Nome completo e matrícula dos integrantes:
- Nome: Anthony Clayton Barros de Jesus Carvalho | Matrícula: 12723110413
- Nome: Anderson Silva Dantas Junior | Matrícula: 1272311567
- Nome: Cauã Souza Conceição | Matrícula: 1272310702
- Nome: Caio Sterphen Barbosa Santos | Matrícula: 12723115470
- Nome: Eros Eloi Alves da Paixão | Matrícula: 12723126200
- Nome: Italo da Conceicao Araujo | Matrícula: 12723114551

2. **Requisitos de Software para Execução da Aplicação**

A aplicação foi desenvolvida com os seguintes requisitos técnicos:
- Linguagem: Java 17
- Gerenciador de dependências: Apache Maven
- Servidor de aplicação: Apache Tomcat 11.0.7
- Banco de dados: SQLite (banco leve baseado em arquivos)
- Conector JDBC: SQLite JDBC
- Frontend: JSP (JavaServer Pages), HTML, CSS e JavaScript puro (vanilla)
- Comunicação cliente-servidor: API REST (não RESTful) baseada em Servlets
- Empacotamento: WAR
- Ambiente opcional de execução: Docker (imagem publicada no Docker Hub)

3. Instruções para Instalação e Execução
A. Executar com Docker (Recomendado)

Pré-requisito:
- Ter o Docker instalado no sistema (Docker Desktop ou CLI).
- Para instalar: https://docs.docker.com/get-docker/

Execução da aplicação via terminal (Docker CLI):

1. Baixe a imagem:
   docker pull cauaconceicao/a3-sdm-imagem
2. Crie e execute um contêiner:
   docker run -d -p 8080:8080 --name api-container cauaconceicao/a3-sdm-imagem
3. Acesse no navegador: http://localhost:8080

4. Execução via Docker Desktop (interface gráfica):
1. Abra o Docker Desktop
2. Vá em 'Images' > 'Pull from Docker Hub'
3. Busque por: cauaconceicao/a3-sdm-imagem
4. Após o download, clique em 'Run'
5. Configure porta 8080 e nomeie como 'api-container'
6. Acesse no navegador: http://localhost:8080

5. Executar Manualmente (Sem Docker)

Pré-requisitos:
- Java 17 (JAVA_HOME)
- Maven (MAVEN_HOME)
- Apache Tomcat 11.0.7
- Git

1. Clone o projeto:
git clone:
https://github.com/CauaConceicao20/Sistema_de_acompanhamento_de_tarefas.git
2. **Compile com Maven:**
mvn clean package
3. Copie o .war para o diretório webapps/ do Tomcat
4. Inicie o Tomcat:
./bin/startup.sh (Linux/Mac)
.\bin\startup.bat (Windows)
5. Acesse:
http://localhost:8080/sistema-de-acompanhamento-de-tarefas-1.0-SNAPSHOT/login

5. Justificativa para a Abordagem de Comunicação Escolhida

Por que utilizamos uma API REST (baseada em Servlets)?
A escolha por uma API REST, ainda que não completamente RESTful, foi técnica e estrategicamente a melhor decisão para este sistema. A seguir estão os principais motivos:

1 - Separação entre interface e lógica de negócio:
A API funciona como uma camada intermediária entre o frontend (JSP/HTML/JS) e a lógica de negócio, possibilitando desacoplamento e favorecendo manutenções futuras ou mudanças na interface.

2 - Facilidade de integração com múltiplos tipos de clientes:
Como o sistema possui três perfis distintos (funcionário, supervisor, gerente), uma API facilita a centralização da lógica e permite que múltiplas interfaces interajam com o mesmo backend, inclusive outras plataformas (como apps ou CLIs no futuro).

3 -Padronização da comunicação:
Usar HTTP com respostas em JSON permite testar as rotas com ferramentas como Postman e Insomnia, além de tornar o comportamento previsível, inclusive para depuração e monitoramento.

4 - Baixa dependência de frameworks:
Ao construir a API usando apenas Servlets, sem usar Spring ou outros frameworks pesados, o sistema permanece didático, controlado e leve, ideal para o propósito educacional e avaliativo.

5 - Escalabilidade e testes:
Separar o backend via API torna o sistema mais testável e mais preparado para evoluções futuras, como autenticação externa ou camadas de segurança mais avançadas.

Por que Java foi a linguagem adequada para este projeto ?

1 - Maturidade da linguagem e robustez:
Java possui suporte maduro a aplicações web, com ecossistema consolidado para aplicações baseadas em Servlets e JSP.

2 - Controle detalhado:
A ausência de frameworks como Spring obrigou a configuração manual de sessões, autenticação, filtros, e manipulação de erros. Isso proporciona aprendizado profundo sobre o funcionamento interno de uma aplicação web.

3 - Integração total com Tomcat, Servlets, JSP e JDBC:
A stack Java EE tradicional utilizada (Servlets + JSP + JDBC) é compatível de forma nativa com o servidor Apache Tomcat, reduzindo sobrecarga e evitando conflitos de versões ou dependências.

4 - Facilidade de conteinerização:
Java, especialmente em conjunto com Tomcat, é altamente compatível com ambientes Docker, facilitando a criação de imagens, testes e deploys.

5 - Alinhamento com o escopo educacional:
O projeto foi planejado para demonstrar o funcionamento da arquitetura MVC tradicional, controle de fluxo HTTP, sessão, autenticação e persistência, tudo em Java puro — o que seria dificultado com frameworks prontos.