

SPD - Relatório Projeto Final

Alunos: Daired Almeida Cruz e Hugo Alves dos Santos

1. Introdução

A gestão eficiente de bibliotecas é um pilar fundamental para o acesso à informação e à cultura. No entanto, muitas instituições ainda dependem de processos manuais ou sistemas legados, que são ineficientes, propensos a erros e inadequados para as expectativas do usuário moderno. A dificuldade em localizar obras, a lentidão no registro de empréstimos e devoluções e a falta de acesso remoto ao acervo são problemas comuns que prejudicam tanto a experiência dos usuários quanto a produtividade dos administradores.

1.1 Descrição do Problema

O desafio central é a modernização do gerenciamento de uma biblioteca. Os principais problemas a serem resolvidos incluem:

- **Ineficiência Operacional:** A ausência de um sistema centralizado para operações de cadastro (CRUD - Create, Read, Update, Delete) de livros, usuários e empréstimos consome tempo e aumenta a chance de falhas humanas.
- **Falta de Acessibilidade:** Os usuários não possuem uma forma prática de consultar a disponibilidade de livros, verificar o status de seus empréstimos ou descobrir novas obras sem se dirigir fisicamente à biblioteca.
- **Silos de Dados:** A arquitetura de muitas bibliotecas consiste em sistemas de persistência de dados distintos e não integrados, como um banco de dados relacional para o gerenciamento interno e outro, NoSQL, para consulta pública. Manter a consistência entre esses bancos é um desafio técnico complexo, que pode levar a informações desatualizadas.

1.2 Motivação

A principal motivação é construir um ecossistema de software coeso, aplicando padrões de arquitetura modernos. O sistema proposto combina uma aplicação desktop para a administração interna com uma API RESTful para consulta pública, garantindo a separação de responsabilidades e a escalabilidade. A implementação de um integrador de dados com JMS para sincronizar um banco relacional SQLite e um banco de documentos MongoDB.

Seção 2 - Plano

2.1 Objetivo Geral

Desenvolver um sistema de gerenciamento de biblioteca multiplataforma, composto por uma aplicação desktop para administração e uma API RESTful para consulta pública. O

objetivo é criar uma solução integrada, eficiente e moderna, que garanta a consistência dos dados entre os diferentes sistemas de persistência (relacional e NoSQL) e ofereça uma experiência de usuário aprimorada tanto para administradores quanto para os leitores da biblioteca.

2.2 Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos serão perseguidos:

- 1. Desenvolver uma Aplicação Desktop (Administração):**
 - Implementar uma interface gráfica intuitiva com **JavaFX**, seguindo o padrão arquitetural **MVC** (Model-View-Controller).
 - Criar funcionalidades completas de **CRUD** (Create, Read, Update, Delete) para as entidades principais: Livro, Autor, Categoria, Usuário, Empréstimo e etc.
 - Utilizar **ORMLite** e **SQLite** para a persistência de dados local, garantindo uma operação leve e eficiente.
 - Publicar eventos de alteração de dados (criação, atualização, exclusão) em um tópico de mensageria para notificar outros componentes do sistema.
- 2. Desenvolver uma API RESTful (Consulta):**
 - Construir uma API utilizando **Spring Boot** para expor endpoints de consulta de livros com seus autores e as resenhas dos usuários.
 - Empregar **MongoDB** como banco de dados NoSQL (orientado a documentos) para otimizar a velocidade e a flexibilidade das consultas.
 - Configurar a API para "escutar" os eventos de alteração vindos da aplicação desktop e atualizar seu próprio banco de dados (MongoDB) de acordo, mantendo a sincronia.
- 3. Implementar um Integrador de Dados:**
 - Garantir a **sincronização assíncrona e desacoplada** entre a aplicação desktop (SQLite) e a API (MongoDB).
 - Adotar uma solução de mensageria para mediar a comunicação, permitindo que os sistemas evoluam de forma independente e garantindo a resiliência do ecossistema.

2.3 Tecnologias e Ferramentas

Para a execução do projeto, serão utilizadas as seguintes tecnologias e ferramentas:

Camada/Componente	Tecnologia/Ferramenta	Finalidade
Aplicação Desktop	JavaFX	Construção da interface gráfica (GUI).
	ORMLite	Mapeamento Objeto-Relacional (ORM) para persistência de dados.
	SQLite	Banco de dados relacional embarcado para o sistema de administração.

API RESTful	Spring Boot	Framework para a criação rápida e robusta de APIs.
	Spring Data MongoDB	Mapeamento Objeto-Documento (ODM) para interação com o MongoDB.
	MongoDB	Banco de dados NoSQL para consultas rápidas e flexíveis na API pública.
Integrador de Entidades	Apache ActiveMQ	(Alternativa ao Redis) Message Broker (intermediário de mensagens) para a comunicação assíncrona entre a aplicação desktop e a API.
	JMS (Java Message Service)	API padrão Java para envio e recebimento de mensagens, utilizada para interagir com o ActiveMQ.
Documentação & Modelagem	Markdown e Javadoc	Documentação do projeto.
	Astah UML	Criação de diagramas UML (Classes, Componentes, Sequência).
Testes	JUnit	Framework para a criação e execução de testes unitários.
Controle de Versão	Git & GitHub	Gerenciamento do código-fonte e colaboração.

Seção 3 - Divisão de Tarefas

3.1 Estratégia de Trabalho

- **Hugo (Foco no Backend e Integração):** Responsável pela API RESTful, o banco de dados MongoDB e a configuração do integrador de mensagens.
- **Daired (Foco no Frontend e Banco de Dados Local):** Responsável pela aplicação desktop em JavaFX, o banco de dados SQLite e a lógica de publicação de mensagens.

3.2 Atribuição de Tarefas (Issues)

Responsável: Hugo (Backend & Integração)

ID da Tarefa	Título	Descrição	Prazo
T01	Configurar Ambiente da API	Inicializar o projeto Spring Boot com dependências (Web, MongoDB, JMS/ActiveMQ).	Dia 1

T02	Modelar Entidades para MongoDB	Criar as classes de domínio (Livro com seus autores e as resenhas dos usuários) para a API.	Dia 1
T03	Implementar Repositórios e Endpoints de Consulta	Desenvolver os repositórios (Spring Data) e os controladores REST para consultar o acervo.	Dia 2
T04	Configurar o Message Broker (ActiveMQ)	Instalar e configurar o ActiveMQ. Criar o consumidor de mensagens (@JmsListener) na API para receber atualizações.	Dia 3
T05	Implementar Lógica de Sincronização	Desenvolver o serviço que processa as mensagens recebidas e atualiza o banco de dados MongoDB.	Dia 4
T06	Testes Unitários da API	Criar testes para os controladores e serviços da API.	Dia 5

Responsável: Daired (Frontend & BD Local)

ID da Tarefa	Título	Descrição	Prazo
T07	Configurar Ambiente Desktop	Inicializar o projeto JavaFX com dependências (ORMLite, SQLite, JMS/ActiveMQ).	Dia 1

T08	Modelar Entidades para SQLite	Criar as classes de domínio (Livro, Autor, Categoria, Usuario, Emprestimo e etc) com anotações ORMLite.	Dia 1
T09	Implementar Camada de Persistência (DAO)	Desenvolver as classes de acesso a dados (DAO) para as operações CRUD no SQLite.	Dia 2
T10	Desenvolver Telas de CRUD (JavaFX)	Criar interfaces gráficas para gerenciar livros, autores e categorias.	Dia 3
T11	Implementar Publicador de Mensagens	Desenvolver o serviço que envia uma mensagem para o ActiveMQ sempre que uma alteração (criar, atualizar, excluir) é realizada.	Dia 4
T12	Testes Unitários do Desktop	Criar testes para as classes de domínio e os DAOs.	Dia 5


3.3 Cronograma do Projeto (5 Dias)

Dia	Atividades Principais	Foco Hugo(Backend)	Foco Daired (Frontend)	Meta do Dia
Dia 1	Setup e Modelagem	(T01) Configurar API Spring Boot. (T02) Modelar entidades MongoDB.	(T07) Configurar App JavaFX. (T08) Modelar entidades SQLite.	Ambientes configurados e modelos de dados definidos.


Dia 2	Desenvolvimento da Persistência	(T03) Implementar repositórios e endpoints de consulta.	(T09) Implementar DAOs para o CRUD no SQLite.	Lógica de acesso a dados implementada em ambos os sistemas.
Dia 3	Desenvolvimento da Interface e Integração Inicial	(T04) Configurar ActiveMQ e o consumidor de mensagens.	(T10) Desenvolver as telas principais do CRUD em JavaFX.	API pronta para receber mensagens e telas do desktop funcionais.
Dia 4	Implementação da Sincronização	(T05) Implementar a lógica que atualiza o MongoDB com base nas mensagens recebidas.	(T11) Implementar o código que envia mensagens após operações de CRUD.	Sincronização de dados entre os sistemas funcionando.
Dia 5	Testes e Documentação	(T06) Testes unitários da API. Revisar documentação da API.	(T12) Testes unitários do desktop. Revisar documentação do desktop.	Projeto testado, documentado e pronto para entrega.

Seção 4 - Modelagem Inicial


4.1 Diagrama de Classes

 classeSPD (2).pdf

4.2 Diagrama de Componentes

 componenteSPD.pdf

4.3 Diagrama de Sequência

 componenteSPD.pdf