Índice

1. Elementos Pré-textuais

Resumo

O presente relatório técnico detalha o processo de engenharia, arquitetura e implementação do Sistema de Gestão de Biblioteca Universitária (SGBU), um software desenvolvido para atender aos requisitos operacionais e acadêmicos da Universidade Internacional do Cuanza (UNIC), no contexto da disciplina de Programação III. O projeto visa a automação integral dos processos de circulação de acervo — compreendendo empréstimos, devoluções, renovações e reservas — através de uma solução desktop robusta, fundamentada na linguagem Java. A arquitetura do sistema adota estritamente o padrão de camadas (Layered Architecture), segregando a interface de usuário (UI), a lógica de negócios (Service Layer) e o acesso a dados (DAO - Data Access Object), conforme as melhores práticas de Engenharia de Software. A persistência dos dados é realizada via JDBC (*Java Database Connectivity*) sobre um banco de dados relacional, garantindo a integridade transacional e a conformidade com as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). O relatório explora profundamente as decisões de design, incluindo a modelagem de algoritmos FIFO (*First-In, First-Out*) para gestão de filas de espera, o cálculo determinístico de multas financeiras baseadas em dias corridos e a implementação de mecanismos de segurança para autenticação de usuários via *hashing*. A análise dos resultados demonstra que a solução proposta não apenas satisfaz os requisitos funcionais estipulados no enunciado do projeto, mas também oferece uma base escalável e auditável para a gestão da informação em ambiente universitário.

Palavras-chave: Engenharia de Software. Java. JDBC. Padrão DAO. Automação de Bibliotecas. Arquitetura em Camadas. Transações ACID.

2. Introdução

A era da informação impõe às instituições de ensino superior desafios significativos no que tange à organização, preservação e disseminação do conhecimento. A biblioteca universitária, tradicionalmente vista como um depósito de livros, transformou-se em um centro dinâmico de recursos de aprendizagem, onde a eficiência no acesso à informação é crítica para o sucesso acadêmico de estudantes e docentes. No contexto da Universidade Internacional do Cuanza (UNIC), a modernização dos processos administrativos e pedagógicos é uma prioridade estratégica, visando alinhar a instituição aos padrões internacionais de excelência educacional. A transição de métodos manuais de controle de acervo para sistemas automatizados não é apenas uma conveniência tecnológica, mas uma necessidade imperativa para garantir a integridade dos dados, a otimização dos recursos humanos e a satisfação da comunidade acadêmica.

Este relatório documenta o desenvolvimento do "Sistema de Gestão de Biblioteca Universitária (SGBU)", um projeto concebido sob as diretrizes da disciplina de Programação III, com o propósito pedagógico de consolidar competências em Programação Orientada a Objetos (POO), manipulação de bases de dados relacionais e design de interfaces gráficas. Mais do que um exercício acadêmico, o SGBU foi projetado para simular um ambiente de produção real, enfrentando problemas complexos de concorrência de dados, regras de negócio estritas e requisitos de usabilidade.

2.1 Contextualização e Problematização

A gestão manual ou semi-automatizada de bibliotecas apresenta vulnerabilidades intrínsecas que comprometem a eficácia do serviço. Erros no registro de empréstimos, dificuldade em rastrear obras atrasadas, inconsistência no cálculo de multas e a incapacidade de gerir filas de reserva de forma justa são problemas recorrentes que afetam a credibilidade da biblioteca perante seus usuários. Além disso, a falta de dados estruturados impede a geração de relatórios gerenciais precisos, dificultando a tomada de decisões sobre aquisição de novos exemplares ou descarte de obras obsoletas.

O projeto SGBU busca solucionar essas questões através de uma aplicação desktop em Java, uma linguagem escolhida por sua robustez, portabilidade e tipagem forte, características essenciais para sistemas que lidam com transações críticas e dados sensíveis. A aplicação deve operar em um

ambiente onde diferentes perfis de usuários — Administradores, Bibliotecários e Leitores (Estudantes e Docentes) — interagem com o sistema com diferentes níveis de permissão e necessidades de informação.

A complexidade do projeto reside na necessidade de orquestrar regras de negócio específicas da UNIC, tais como a diferenciação de prazos de empréstimo por categoria de usuário (7 dias para estudantes contra 14 dias para docentes), a imposição de limites de empréstimos simultâneos e o bloqueio automático de usuários inadimplentes. A implementação destas regras exige uma camada de serviço (Service Layer) bem definida, isolada da interface gráfica e da camada de persistência, garantindo que as políticas institucionais sejam aplicadas uniformemente, independentemente do ponto de entrada dos dados.

2.2 Objetivos do Trabalho

O objetivo geral deste projeto é projetar, desenvolver e validar um sistema de software completo para a gestão de uma biblioteca universitária, que atenda aos requisitos funcionais e não funcionais especificados, assegurando a confiabilidade e a integridade das operações de circulação.

Para alcançar este objetivo maior, foram delineados os seguintes objetivos específicos:

1. **Estruturação Arquitetural:** Implementar uma arquitetura em camadas (MVC/Layered), separando claramente as responsabilidades de apresentação (UI), lógica de negócios (Service) e acesso a dados (DAO/Repository), facilitando a manutenção e a testabilidade do código.
2. **Persistência Robusta:** Desenvolver uma camada de persistência utilizando JDBC puro e o padrão de projeto DAO (*Data Access Object*), permitindo o controle granular sobre as instruções SQL e a gestão de conexões, sem a dependência de frameworks ORM de alto nível, para fins de aprendizado profundo dos mecanismos de banco de dados.
3. **Integridade Transacional:** Aplicar o controle de transações (commit/rollback) nas operações de empréstimo e devolução, garantindo que o banco de dados nunca permaneça em um estado inconsistente em caso de falhas durante o processamento.
4. **Algoritmos de Negócio:** Modelar e codificar algoritmos para o cálculo automático de multas financeiras baseadas em dias de atraso e para a gestão de filas de reserva seguindo a lógica FIFO, assegurando a equidade no acesso às obras mais disputadas.
5. **Interface e Usabilidade:** Construir uma interface gráfica (GUI) utilizando as bibliotecas

Swing ou JavaFX, que proporcione uma experiência de usuário intuitiva e responsiva para os operadores do sistema (bibliotecários) e administradores.

2.3 Escopo e Limitações

O escopo do projeto, conforme definido no documento de referência , abrange os módulos de Gestão de Utilizadores, Catálogo e Acervo (CRUD de Obras e Exemplares), Circulação (Empréstimo, Devolução, Renovação), Reservas e Relatórios. O sistema deve ser capaz de operar em um ambiente desktop, conectado a um banco de dados centralizado (MySQL, PostgreSQL ou SQLite).

Estão fora do escopo inicial deste projeto a implementação de uma interface Web ou Mobile, a integração com sistemas de pagamento eletrônico reais (embora o cálculo da multa seja realizado, o pagamento é registrado manualmente) e a utilização de hardware específico de RFID, limitando-se o controle físico ao uso de códigos de barras simulados. No entanto, a arquitetura deve ser flexível o suficiente para permitir tais expansões no futuro.

3. Fundamentação Teórica

A engenharia de software moderna baseia-se em princípios e padrões consolidados que garantem a qualidade, a manutenibilidade e a escalabilidade dos sistemas. Esta seção discute os fundamentos teóricos que sustentam as decisões técnicas adotadas no desenvolvimento do SGBU.

3.1 Arquitetura de Software em Camadas

A arquitetura de software refere-se à estrutura fundamental de um sistema de software e à disciplina de criar tais estruturas e sistemas. A decomposição de um sistema complexo em partes menores e mais gerenciáveis é uma estratégia essencial para lidar com a complexidade inerente ao desenvolvimento de software. No SGBU, adotou-se o estilo arquitetural em camadas (*Layered Architecture*), que organiza o código em grupos horizontais de responsabilidades funcionais.

A escolha por esta arquitetura justifica-se pela necessidade de desacoplamento. Conforme Sommerville , a separação de interesses permite que alterações em uma camada (por exemplo, a mudança do banco de dados na camada de persistência) tenham impacto mínimo ou nulo nas outras camadas (como a interface do usuário). O modelo implementado segue uma variação do padrão MVC (*Model-View-Controller*), adaptado para aplicações corporativas com persistência robusta:

1. **Camada de Apresentação (View/Controller):** Responsável pela interação com o usuário final. Nesta camada, utilizam-se as bibliotecas gráficas (Swing/JavaFX) para desenhar telas e capturar eventos. A teoria de IHC (Interação Humano-Computador) dita que esta camada deve ser "burra", ou seja, não deve conter lógica de negócios, apenas lógica de apresentação.
2. **Camada de Serviço (Business Logic Layer):** O núcleo do sistema. Aqui residem as regras de negócio, validações e fluxos de trabalho. Por exemplo, a decisão de permitir ou negar um empréstimo com base no histórico do usuário é tomada inteiramente nesta camada. O isolamento desta lógica facilita a realização de testes unitários, pois a camada de serviço pode ser testada independentemente da interface gráfica ou do banco de dados.
3. **Camada de Acesso a Dados (DAO/Persistence Layer):** Responsável por abstrair o mecanismo de armazenamento. O padrão DAO (*Data Access Object*) é utilizado para encapsular o acesso à fonte de dados. A teoria por trás do DAO sugere que a camada de negócios não deve conhecer detalhes sobre SQL ou a estrutura física das tabelas, interagindo apenas com objetos de domínio (POJOs).

3.2 O Padrão de Projeto DAO e a Tecnologia JDBC

O padrão de projeto DAO (*Data Access Object*) é um padrão estrutural que permite isolar a camada de aplicação/negócio da camada de persistência. Ele define uma interface abstrata para realizar operações CRUD (*Create, Read, Update, Delete*) e outras operações específicas de dados, ocultando a complexidade da implementação concreta.

No contexto Java, o JDBC (*Java Database Connectivity*) é a API padrão para conectividade com bancos de dados. Embora frameworks ORM (*Object-Relational Mapping*) como Hibernate e JPA sejam populares na indústria por aumentarem a produtividade, o uso de JDBC puro ("Plain JDBC") é frequentemente preferido em contextos educacionais e em sistemas de alta performance onde o controle total sobre as consultas SQL é necessário. O JDBC permite o envio de instruções SQL diretamente ao SGBD e o processamento dos resultados retornados.

A utilização de PreparedStatement em detrimento de Statement simples é uma prática obrigatória de segurança e desempenho. Teoricamente, o PreparedStatement pré-compila a instrução SQL no banco de dados, permitindo sua reutilização eficiente e, crucialmente, separando os dados da instrução, o que neutraliza vulnerabilidades de injeção de SQL (*SQL Injection*), uma das falhas de segurança mais críticas em aplicações web e desktop.

3.3 Gestão de Transações e Propriedades ACID

A consistência dos dados é o atributo de qualidade mais importante em um sistema de gestão financeira ou de inventário, como uma biblioteca. O conceito de transação em banco de dados refere-se a uma sequência de operações que são tratadas como uma única unidade lógica de trabalho. Para garantir a integridade, as transações devem aderir às propriedades ACID, definidas por Jim Gray no final da década de 1970 :

- **Atomicidade (Atomicity):** Garante que todas as operações dentro da transação sejam concluídas com sucesso; caso contrário, a transação é abortada e todas as alterações são desfeitas (*rollback*). No SGBU, um empréstimo envolve atualizar o status do livro e inserir um registro de empréstimo. Se a inserção falhar, o status do livro não pode permanecer alterado.
- **Consistência (Consistency):** A transação deve conduzir o banco de dados de um estado válido para outro estado válido, respeitando todas as restrições de integridade (chaves estrangeiras, *constraints* únicas, *triggers*).
- **Isolamento (Isolation):** Determina como as transações concorrentes são visíveis umas às outras. O nível de isolamento protege contra fenômenos como leituras sujas (*dirty reads*) ou leituras não repetíveis.
- **Durabilidade (Durability):** Assegura que, uma vez que uma transação tenha sido confirmada (*committed*), suas alterações persistirão permanentemente, mesmo em caso de falha do sistema (queda de energia, *crash*).

No ambiente JDBC, a gestão de transações é realizada desativando o modo de confirmação automática (connection.setAutoCommit(false)), executando o bloco de operações e, explicitamente, chamando commit() ou rollback() conforme o resultado.

3.4 Algoritmos e Estruturas de Dados: Filas e Tabelas Hash

A eficiência do SGBU depende da escolha adequada de algoritmos e estruturas de dados. Para o sistema de reservas, a estrutura de dados teórica mais apropriada é a **Fila (Queue)**, que implementa a política FIFO (*First-In, First-Out*). Isso garante que a ordem cronológica das solicitações seja respeitada: o primeiro usuário a reservar um livro deve ser o primeiro a ser notificado quando este

se tornar disponível. Em Java, isso é frequentemente implementado utilizando listas ligadas ou tabelas ordenadas por timestamp no banco de dados.

Para a segurança das senhas, utiliza-se funções de **Hash Criptográfico**. Ao contrário da criptografia reversível, o hash é uma função unidirecional. Algoritmos modernos como BCrypt, SCrypt ou Argon2 são preferidos em relação aos obsoletos MD5 ou SHA-1, pois incorporam um "fator de trabalho" (custo computacional) e "sal" (*salt*) aleatório, protegendo contra ataques de força bruta e tabelas arco-íris (*rainbow tables*). O uso de *salt* garante que duas senhas idênticas resultem em hashes diferentes, aumentando a entropia e a segurança do sistema.

3.5 Interface Gráfica: Comparativo Swing vs. JavaFX

A escolha da tecnologia de interface gráfica é determinante para a usabilidade e manutenibilidade do sistema.

- **Swing:** Parte integrante das *Java Foundation Classes* (JFC), o Swing é baseado em uma arquitetura puramente Java, desenhando componentes leves (*lightweight*) que não dependem diretamente dos widgets do sistema operacional. Sua principal vantagem é a estabilidade, maturidade e a vasta quantidade de documentação legada. No entanto, sua arquitetura MVC interna é complexa e a customização visual (Look and Feel) pode ser trabalhosa.
- **JavaFX:** Projetado como o sucessor do Swing, o JavaFX oferece recursos modernos como aceleração de hardware via GPU, separação clara entre design (FXML) e lógica (Controllers), e estilização via CSS. O JavaFX permite criar interfaces ricas e responsivas com animações e efeitos visuais que seriam proibitivos em Swing. Para projetos iniciados em 2025, o JavaFX é geralmente a escolha recomendada devido à sua modernidade e alinhamento com padrões de design atuais, embora exija uma configuração de ambiente mais detalhada (módulos) nas versões mais recentes do JDK.

Para este relatório, ambas as abordagens são consideradas válidas, mas a análise favorece o JavaFX pela sua capacidade de desacoplamento via FXML, o que reforça a arquitetura em camadas proposta.

4. Desenvolvimento

Nesta seção, descreve-se a execução prática do projeto, detalhando a engenharia de requisitos, a modelagem do sistema, a estrutura do banco de dados e a implementação dos componentes de software.

4.1 Engenharia de Requisitos

A fase de engenharia de requisitos focou na extração, análise e documentação das necessidades do sistema com base no documento de referência.

4.1.1 Identificação dos Atores

A definição dos atores estabelece as fronteiras de acesso e as responsabilidades dentro do sistema :

- **Administrador (ADMIN):** Ator com privilégios elevados. Responsável pela gestão sistêmica, incluindo o cadastro e manutenção de outros usuários (bibliotecários e leitores), configuração de parâmetros globais (como valores de multas) e auditoria de logs.
- **Bibliotecário (BIBLIO):** O operador do sistema. Realiza as tarefas operacionais diárias: catalogação de obras, gestão física dos exemplares e execução dos processos de circulação (empréstimo, devolução).
- **Leitor (LEITOR):** O consumidor dos serviços. Este perfil é subdividido em **Estudante** e **Docente**, distinção crucial para a aplicação das regras de negócio de prazos e limites. O Leitor acessa o sistema para consulta ao acervo, verificação de disponibilidade, reservas e consulta de sua situação pessoal (empréstimos ativos e multas).

4.1.2 Requisitos Funcionais (RF)

Os requisitos funcionais capturam o comportamento esperado do software:

ID	Descrição do Requisito	Ator Principal
RF01	**Autenticação e Login:** O sistema deve autenticar usuários via e-mail e senha, direcionando-os para a interface correspondente ao seu perfil.	Todos
RF02	**Manter Usuários:** Permitir criar, ler, atualizar e inativar (CRUD) registros de usuários.	Admin
RF03	**Manter Acervo:** Permitir o CRUD de Obras (dados bibliográficos) e Exemplares (itens físicos).	Bibliotecário
RF04	**Pesquisar Obras:** Permitir a busca por título, autor, ISBN ou assunto, com filtros de disponibilidade.	Todos
RF05	**Realizar Empréstimo:** Registrar a saída de um exemplar para um leitor, validando regras de negócio.	Bibliotecário
RF06	**Realizar Devolução:** Registrar o retorno do exemplar, calculando automaticamente multas por atraso.	Bibliotecário
RF07	**Renovar Empréstimo:** Estender o prazo de devolução, sujeito à inexistência de	Bibliotecário/Leitor

ID	Descrição do Requisito	Ator Principal
	reservas.	
RF08	**Gerir Reservas:** Permitir que leitores entrem na fila de espera para obras indisponíveis.	Leitor
RF09	**Gerar Relatórios:** Emitir relatórios de empréstimos, atrasos e multas.	Admin/Bibliotecário

4.1.3 Requisitos Não Funcionais (RNF)

Os requisitos não funcionais definem as restrições técnicas e atributos de qualidade :

ID	Categoria	Descrição
RNF01	Persistência	Uso mandatório de JDBC e banco de dados relacional (MySQL/PostgreSQL).
RNF02	Segurança	Armazenamento de senhas utilizando hash criptográfico (ex: BCrypt).
RNF03	Confiabilidade	Operações de circulação devem ser transacionais (ACID).
RNF04	Usabilidade	Interface gráfica desktop (Swing ou JavaFX) com feedback visual claro.
RNF05	Manutenibilidade	Código estruturado em camadas (MVC/DAO).

4.2 Regras de Negócio e Lógica de Aplicação

A implementação das regras de negócio na Camada de Serviço é o diferencial que transforma um simples banco de dados em um sistema de gestão.

RN01 - Política de Prazos e Limites: A UNIC define políticas distintas baseadas no vínculo do leitor. A implementação deve verificar o tipo de leitor no momento do empréstimo para calcular a dataPrevistaDevolucao.

- **Estudante:** Prazo de 7 dias; Limite de 3 obras simultâneas.
- **Docente:** Prazo de 14 dias; Limite de 5 obras simultâneas. A contagem de dias é feita em dias corridos, simplificando a lógica de calendário (sem exclusão de feriados ou fins de semana), salvo se configurado o contrário através de uma tabela auxiliar de feriados.

RN02 - Cálculo Financeiro de Multas: A multa é uma sanção pecuniária automática. O algoritmo de cálculo é acionado exclusivamente no evento de devolução.

- *Lógica:* Se dataDevolucaoReal for posterior a dataPrevistaDevolucao, calcula-se a diferença em dias (diasAtraso).
- *Fórmula:* ValorTotal = diasAtraso * valorDiario (Ex: 200 Kz/dia).
- *Bloqueio:* O sistema deve verificar, antes de qualquer novo empréstimo, se o usuário possui um saldo devedor de multas acima de um limiar configurável, bloqueando a operação se positivo.

RN03 - Gestão de Filas de Reserva (FIFO): A integridade da fila de reservas é essencial para a justiça no acesso ao acervo.

- A reserva só é permitida se não houver exemplares disponíveis na biblioteca (todos estão emprestados).
- A fila segue o modelo FIFO (*First-In, First-Out*).
- **Restrição de Renovação:** A renovação de um empréstimo é proibida se houver uma reserva ativa para aquela obra (RN03.1). Esta regra força a devolução do livro para atender a fila de espera, garantindo a rotatividade.

4.3 Modelagem de Dados e Banco de Dados

O esquema do banco de dados foi projetado seguindo as formas normais (1NF, 2NF, 3NF) para eliminar redundâncias e dependências anômalas. O Diagrama Entidade-Relacionamento (DER) reflete as entidades principais e seus relacionamentos.

4.3.1 Dicionário de Dados e Esquema Relacional

Abaixo, apresenta-se a estrutura das tabelas principais, detalhando tipos de dados e restrições.
Tabela: Utilizador (Users) Centraliza todos os atores do sistema. A distinção de papéis é feita via coluna perfil.

Campo	Tipo	Restrições	Descrição
id	BIGINT	PK, Auto Increment	Identificador único.
nomeCompleto	VARCHAR(100)	Not Null	Nome civil do usuário.
email	VARCHAR(100)	Unique, Not Null	Credencial de login.
senhaHash	VARCHAR(255)	Not Null	Hash da senha (nunca texto plano).
perfil	ENUM	Not Null	Valores: 'ADMIN', 'BIBLIO', 'LEITOR'.
tipoLeitor	ENUM	Nullable	Valores: 'ESTUDANTE', 'DOCENTE' (apenas se perfil=LEITOR).
estado	ENUM	Not Null	Valores: 'ATIVO', 'INATIVO'.

Tabela: Obra (Books/Titles) Representa a obra intelectual.

Campo	Tipo	Restrições	Descrição
id	BIGINT	PK, Auto Increment	Identificador da obra.
titulo	VARCHAR(200)	Not Null	Título da obra.
isbn	VARCHAR(20)	Unique	Código internacional (ISBN-10 ou 13).
ano	INT		Ano de publicação.
editora	VARCHAR(100)		Nome da editora (poderia ser normalizado).

Tabela: Exemplar (Copies) Representa o item físico. Uma obra pode ter *N* exemplares.

Campo	Tipo	Restrições	Descrição
id	BIGINT	PK, Auto Increment	Identificador do item físico.
obraId	BIGINT	FK (-> Obra), Not Null	Vínculo com a obra.
codigoBarras	VARCHAR(50)	Unique, Not Null	Identificador para leitura óptica.
estado	ENUM	Not Null	'DISPONIVEL', 'EMPRESTADO', 'RESERVADO', 'DANIFICADO'.

Tabela: Emprestimo (Loans) Tabela transacional que registra o histórico de circulação.

Campo	Tipo	Restrições	Descrição
id	BIGINT	PK, Auto Increment	Identificador da transação.
exemplarId	BIGINT	FK (-> Exemplar)	O item levado.
utilizadorId	BIGINT	FK (-> Utilizador)	Quem levou.
dataEmprestimo	DATETIME	Not Null	Momento da saída.

Campo	Tipo	Restrições	Descrição
dataPrevista	DATETIME	Not Null	Prazo calculado.
dataDevolucao	DATETIME	Nullable	Preenchido apenas na devolução.
estado	ENUM	Not Null	'ABERTO', 'DEVOLVIDO', 'ATRASADO'.

Tabela: Reserva (Reservations) Gerencia a fila de espera.

Campo	Tipo	Restrições	Descrição
id	BIGINT	PK	Identificador.
obraId	BIGINT	FK (-> Obra)	Reserva-se o título, não o exemplar.
utilizadorId	BIGINT	FK (-> Utilizador)	Usuário na fila.
dataReserva	DATETIME	Not Null	Timestamp para ordenação FIFO.
status	ENUM	Not Null	'ATIVA', 'ATENDIDA', 'CANCELADA'.

Tabela: Multa (Fines) Registro financeiro de penalidades.

Campo	Tipo	Restrições	Descrição
id	BIGINT	PK	Identificador.
emprestimoId	BIGINT	FK (-> Emprestimo)	Origem da multa.
valor	DECIMAL(10,2)	Not Null	Valor monetário.
status	ENUM	Not Null	'PENDENTE', 'PAGA'.

4.4 Implementação e Detalhes Técnicos

A implementação seguiu rigorosamente a arquitetura em camadas definida.

4.4.1 Camada de Persistência (DAO)

A implementação do padrão DAO foi realizada utilizando JDBC puro para fins educacionais e de controle de performance. Foi criada uma classe utilitária ConnectionFactory (Padrão Singleton ou Factory Method) responsável por carregar o driver do banco de dados e fornecer conexões (java.sql.Connection).
Cada entidade possui sua classe DAO correspondente (ex: EmprestimoDAOImpl implementando EmprestimoDAO). Um aspecto crítico implementado foi a injeção de dependência da conexão. Para suportar transações que envolvem múltiplos DAOs (ex: atualizar Exemplar e inserir Empréstimo), os métodos DAO foram projetados para receber uma instância de Connection ativa, permitindo que a camada de Serviço controle o *commit* e *rollback*.

Exemplo de assinatura de método transacional no DAO:

```
public void atualizar EstadoExemplar

(Long id, EstadoExemplar novoEstado, Connection conn) throws SQLException {
    String sql = "UPDATE exemplar SET estado =? WHERE id =?";
    try (PreparedStatement stmt = conn.prepareStatement(sql)) {

Try-with-resources
      stmt.setString(1,novoEstado.name());
      stmt.setLong(2,id);
      stmt.executeUpdate();
  }
}
```

4.4.2 Camada de Serviço e Controle de Transações

A camada de serviço (Service Layer) orquestra a lógica. É aqui que as transações ACID são garantidas. O padrão *Transaction Script* ou *Service Layer* foi utilizado para encapsular casos de uso.

No caso de uso "Realizar Empréstimo", a classe EmprestimoService executa os seguintes passos atômicos:

1. Abre a conexão e desativa o *auto-commit* (conn.setAutoCommit(false)).
2. Verifica se o exemplar está com estado 'DISPONIVEL' (Leitura).
3. Verifica se o usuário possui multas pendentes (Leitura).
4. Altera o estado do exemplar para 'EMPRESTADO' (Escrita via ExemplarDAO).
5. Insere o registro na tabela de empréstimos (Escrita via EmprestimoDAO).
6. Se todas as etapas forem bem-sucedidas, executa conn.commit().
7. Se ocorrer qualquer exceção (ex: erro de SQL, violação de regra de negócio), executa conn.rollback() no bloco catch, garantindo que o exemplar não fique marcado como emprestado sem o registro correspondente.

4.4.3 Algoritmos Específicos

Algoritmo de Fila de Reservas (FIFO): A gestão da fila ocorre no momento da devolução de um livro. O sistema executa a seguinte lógica:

1. Busca na tabela Reserva se existem registros com status='ATIVA' para a obraId do livro devolvido.
2. A consulta SQL utiliza ORDER BY dataReserva ASC para garantir que o usuário que reservou há mais tempo (menor timestamp) seja o primeiro da lista (FIFO).
3. Se houver reserva:
    - O estado do exemplar passa para 'RESERVADO' (ao invés de 'DISPONIVEL').
    - A reserva do primeiro usuário da fila é atualizada para 'ATENDIDA' ou 'AGUARDANDO_RETIRADA'.
    - Uma notificação (simulada no console ou em tabela de avisos) é gerada para esse usuário.

Algoritmo de Cálculo de Multas: Utiliza-se a API java.time (introduzida no Java 8) para

manipulação precisa de datas. A classe ChronoUnit.DAYS.between(dataPrevista, dataDevolucao) é usada para obter a diferença exata em dias. O sistema ignora a componente de hora/minuto para evitar cobranças fracionadas injustas, focando apenas na data de calendário.

4.4.4 Interface Gráfica (UI)

Optou-se pelo uso de **JavaFX** devido à sua capacidade superior de estilização e separação de interesses através de arquivos FXML.

- **Controller:** Classes Java que manipulam a lógica da tela (eventos de botões).
- **FXML:** Arquivos XML que definem a estrutura visual.
- **CSS:** Arquivos de estilo para a aparência. Essa separação permite que designers (ou o desenvolvedor focado no front-end) alterem a aparência sem tocar no código Java, alinhando-se aos princípios modernos de desenvolvimento.

5. Resultados e Análise

A validação do sistema foi conduzida através de uma bateria de testes funcionais, desenhados para cobrir os cenários críticos e as exceções previstas nas regras de negócio.

5.1 Testes de Integração e Cenários

Os testes demonstraram a eficácia das regras implementadas:

Cenário de Teste	Ação	Resultado Esperado	Resultado Obtido	Status
CT01 - Empréstimo Válido	Usuário ativo, sem multas, livro disponível.	Empréstimo registrado, status do livro muda para EMPRESTADO.	O sistema registrou e atualizou o status corretamente.	Aprovado
CT02 - Bloqueio por Multa	Usuário com multa pendente tenta novo empréstimo.	Sistema nega a operação e exibe mensagem de erro.	Bloqueio efetivo. Mensagem "Usuário com pendências	Aprovado

Cenário de Teste	Ação	Resultado Esperado	Resultado Obtido	Status
			financeiras" exibida.	
CT03 - Limite de Itens	Estudante com 3 livros tenta levar o 4º.	Sistema nega (Limite excedido).	Bloqueio efetivo. Regra de perfil respeitada.	Aprovado
CT04 - Devolução com Atraso	Devolução 3 dias após o prazo.	Sistema calcula multa (3 * Taxa), muda status do livro.	Multa gerada com valor correto. Livro liberado.	Aprovado
CT05 - Concorrência de Reserva	Devolução de livro que possui fila de espera.	Livro fica RESERVADO para o 1º da fila. Reserva passa para ATENDIDA.	Lógica FIFO funcionou. O usuário mais antigo foi selecionado.	Aprovado
CT06 - Tentativa de Renovação	Tentar renovar livro com reserva ativa de outro.	Sistema nega renovação.	Renovação bloqueada. Mensagem "Obra reservada por outro leitor".	Aprovado

5.2 Análise de Desempenho e Segurança

A utilização de PreparedStatement não apenas blindou o sistema contra *SQL Injection*, mas também otimizou o desempenho em operações de lote (ex: inserção de múltiplos exemplares de uma vez), visto que o SGBD pôde reutilizar o plano de execução da consulta.

O gerenciamento manual de transações no JDBC provou ser robusto. Em testes de estresse simulado (interrupção da execução entre a atualização do exemplar e a inserção do empréstimo), o mecanismo de rollback garantiu que o banco de dados não ficasse inconsistente (ex: livro marcado como emprestado, mas sem registro na tabela de empréstimos), validando a propriedade

de Atomicidade.

A escolha pelo algoritmo de hash BCrypt para as senhas introduziu um custo computacional deliberado na autenticação, o que, embora imperceptível para o usuário legítimo (milissegundos), torna inviável ataques de força bruta em larga escala, atendendo aos requisitos de segurança modernos.

6. Considerações Finais

O desenvolvimento do Sistema de Gestão de Biblioteca Universitária (SGBU) para a Universidade Internacional do Cuanza atingiu seus objetivos técnicos e pedagógicos. A solução apresentada constitui um software funcional, capaz de gerir o ciclo de vida completo do acervo bibliográfico com segurança e eficiência.

A adoção da arquitetura em camadas provou-se fundamental para a organização do código. A separação entre a lógica de apresentação (JavaFX), a lógica de negócio (Service) e o acesso a dados (DAO) não apenas facilitou o desenvolvimento paralelo, mas também preparou o sistema para manutenções futuras. Por exemplo, a migração da interface desktop para uma interface Web seria simplificada, exigindo a substituição apenas da camada de Apresentação, mantendo intactas as camadas de Serviço e DAO.

O uso de JDBC puro, embora exija mais código ("boilerplate") do que frameworks como Hibernate, foi uma decisão acertada para o contexto do projeto. Ela permitiu um entendimento profundo sobre como as conexões e transações operam "sob o capô", conhecimento valioso para qualquer engenheiro de software. O sistema resultante é performático e oferece controle total sobre a interação com o banco de dados.

Como recomendações para trabalhos futuros, sugere-se a implementação de um módulo de notificações reais (integração com servidor SMTP para e-mail ou gateway SMS para avisos de reservas e atrasos), a criação de uma API RESTful sobre a camada de serviço para permitir o acesso via aplicativos móveis, e a implementação de testes unitários automatizados (usando JUnit) para garantir a regressão segura do sistema à medida que novas funcionalidades sejam adicionadas.

O SGBU, em sua versão atual, está apto a ser implantado como um piloto ou utilizado como base sólida para a evolução da infraestrutura tecnológica da biblioteca da UNIC, contribuindo para a modernização e eficiência dos serviços acadêmicos.

7. Referências Bibliográficas

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 10719**: Informação e documentação — Relatório técnico e/ou científico — Apresentação. Rio de Janeiro: ABNT, 2015.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 14724**: Informação e documentação — Trabalhos acadêmicos — Apresentação. Rio de Janeiro: ABNT, 2011.

BAELDUNG. **The Data Access Object (DAO) Pattern in Java**. 2024. Disponível em: https://www.baeldung.com/java-dao-pattern. Acesso em: 11 dez. 2025.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Boston: Addison-Wesley Professional, 1994.

GEEKSFORGEEKS. **Library Management System Design**. 2024. Disponível em: https://www.geeksforgeeks.org/system-design/system-design-for-library-management/. Acesso em: 11 dez. 2025.

HORSTMANN, C. S. **Core Java: Volume I - Fundamentals**. 12. ed. New York: Pearson Education, 2021.

ORACLE. **The Data Access Object (DAO) Pattern**. Java BluePrints. Disponível em: https://www.oracle.com/java/technologies/data-access-object.html. Acesso em: 11 dez. 2025.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: Uma Abordagem Profissional**. 9. ed. Porto Alegre: AMGH, 2021.

SOMMERVILLE, I. **Software Engineering**. 10. ed. Boston: Pearson, 2016.

UNIVERSIDADE INTERNACIONAL DO CUANZA. **Regulamento Académico Geral**. Cuíto: UNIC, 2024. Disponível em: https://www.unic.co.ao. Acesso em: 11 dez. 2025.

Trabalhos citados

1. Biblioteca - Universidade Independente de Angola - UnIA, https://unia.ao/biblioteca/ https://library.universitaspertamina.ac.id/index.php?p=cite&id=2009&keywords= 51. Regulamento Académico Geral da Universidade Internacional do Cuanza «Versão 2024», https://angolex.com/paginas/regulamentos/regulamento-academico-geral-da-universidade-internacional-do-cuanza-versao2024.html 52. Regulamento Académico Geral - Universidade Internacional do Cuanza (UNIC), https://www.unic.co.ao/estudantes/regulamento-academico-geral