1. Introdução

Este documento tem como finalidade apresentar a documentação básica do sistema de Biblioteca desenvolvido ao final da disciplina de Engenharia de Software II como requisito parcial para a obtenção de nota. A documentação abrange os principais *artefatos* necessários para o entendimento técnico da solução implementada na aplicação, está por sua vez pretende auxiliar as atividades corriqueiras desempenhadas no ambiente de biblioteca, manipulando informações referentes a alunos, livros, empréstimos existentes, entre outros.

2. Justificativa e objetivos

A chegada e a consolidação da informática na sociedade ao longo do tempo têm gradativamente substituído o papel da mídia tradicional na distribuição de informações em larga escala. Mais do que isso, a informática tornou-se o pilar de um novo estilo de vida característico da modernidade, influenciando padrões de comportamento relacionados ao lazer, trabalho, consumo, educação e ao mercado profissional. Com ela, surgiu também um senso coletivo de “aceleração” do tempo, impulsionado pela instantaneidade proporcionada pelos novos meios de comunicação, como mensagens e e-mails. Essas mudanças configuraram um ambiente dinâmico e em constante transformação, exigindo que indivíduos e organizações se adaptem a um ritmo acelerado e lidem com um alto grau de complexidade (LIMA; PINTO; LAIA, 2002).

Davenport (2002) descreve o gerenciamento informacional como um processo do qual compreende um conjunto estruturado de atividades nas quais incluem o modo como as empresas obtêm, distribuem, usam da informação e do conhecimento. As bibliotecas como unidades organizacionais ativas que lidam com a gestão informacional sofrem interferências diárias de ações de cunho cultural, social ou mesmo econômico frente aos seus processos trabalho. Logo se faz imprescindível a sua adequação perante a essas transformações para a garantia da qualidade no serviço prestado e portanto, o alcance da satisfação por parte do usuário final. O que se sucede então é um movimento colaborativo entre bibliotecários e analistas de sistemas para a modernização das unidades por meio da implantação da tecnologia da informação (DO AMARAL; ZAFALON, 2009).

Como resultado dessa modernização tarefas antes manuais passam a ser automatizadas, ou seja, desempenhadas seja via *hardware* ou *software*. Nesse cenário a automatização de atividades proporciona, dentre outros benefícios, um atendimento eficiente e eficaz ao usuário, a redução do tempo gasto em determinadas obrigações, a otimização destas mesmas, a ampliação do número de clientes atendidos, o auxílio na aquisição de novos produtos, uma maior precisão na organização dos conteúdos e principalmente atendimento dos usuários em um curto espaço de tempo. Existem ainda ganhos por parte dos profissionais que atuam nesse espaço ao uniformizar certos aspectos inerentes ao seu trabalho (geração de relatórios, atendimento) e evitar esforços desgastantes e repetitivos tais como a catalogação e a busca por títulos. “Os bibliotecários gastam a maior parte do seu tempo, energia e recursos na descrição bibliográfica, análise de conteúdo e na busca do significado ou importância desses conteúdos” (WHITE 1993, p. 259).

Estas questões motivaram o desenvolvimento de um sistema de biblioteca, sendo que para a sua elaboração utilizou-se o paradigma orientado a objetos no qual organiza as funcionalidades em módulos específicos, favorecendo deste modo a evolução e futuras manutenções no sistema. O código está escrito na linguagem de programação Java, sendo que para a construção das interfaces foi adotada a tecnologia JavaFX por permitir uma interação com o usuário de forma estruturada. Para a persistência das informações recorreu-se ao banco de dados MySQL. A integração entre esses componentes ocorre por meio do JDBC (Java Database Connectivity), permitindo operações seguras e consistentes de manipulação dos dados.

3. Metodologia

Larman (2005) enuncia que a ferramenta crucial no desenvolvimento de software não é nenhuma tecnologia popular, metodologia ou recurso de modelagem utilizado no projeto, tal como a UML, mas sim uma mente bem educada nos princípios de projeto.

Um software é constituído a partir de uma coleção de objetos de software, os quais cada um, assim como uma célula em um sistema biológico, não possui o conhecimento dos processos particulares dos demais, no entanto isso não impede que haja uma comunicação entre estes elementos menores afim de desempenharem tarefas complexas. Objetos possuem comportamentos estritamente limitados àqueles projetados para eles, é somente quando há a interação entre vários deles que se alcança um comportamento dinâmico do sistema, com cada um exercendo seu papel definido. Os objetos de software exibem informações, coordenam atividades, calculam e se conectam a serviços. Por poderem ser associados para trabalharem conjunto, pode advir dessa agregação sistemas altamente complexos, uma maneira de gerenciar essa complexidade é atribuir *responsabilidades* bem definidas a cada objeto (WIRFS-BROCK; MCKEAN, 2002). O projeto guiado por responsabilidades (PGR) raciocina sobre o projeto de objetos de software em termos de *responsabilidades, papéis* e *colaborações.*

Neste contexto, a abordagem GRASP (General responsibility assignment software patterns) se mostra relevante por fornecer um conjunto de diretrizes para atribuir responsabilidade a classes e objetos dentro de um projeto orientado a objetos, permitindo raciocinar sobre o mesmo de modo metódico, racional e explicável. Entre os princípios que o mesmo propõe destacam-se a **alta coesão** e o **baixo acoplamento**, por se manifestarem como indicativos de qualidade do projeto, refletindo a capacidade do *design* de ser modular, flexível e de fácil manutenção.

A alta coesão pode ser descrita uma qualidade básica, que corresponde informalmente, ao quão relacionadas funcionalmente estão as operações de um elemento de software, ou seja o quanto seus métodos refletem a responsabilidade definida para o mesmo. Ela também mede o quão sobrecarregado um elemento se encontra (mais métodos que o necessário). Conforme um componente de software passa a cobrir responsabilidades de diferentes áreas do sistema sua coesão abaixa, com pouca coesão há muita colaboração com outros artefatos, o que resulta em um alto acoplamento. Já o baixo acoplamento por sua vez pode ser caracterizado como uma medida de quão fortemente um elemento está conectado a, tem conhecimento de, ou depende de outros elementos. Um acoplamento existe em um cenário onde uma classe A herda ou estende o comportamento de uma classe B, ou mesmo quando A faz uso dos serviços de B. Um acoplamento alto, possivelmente advindo de uma atribuição de responsabilidade erronia gera dependências desnecessárias entre classes o que a longo prazo resulta em percas de tempo, esforço e falhas na correção do software (LARMAN, 2005).

Nesse sentido buscou-se desenvolver um sistema cuja as responsabilidades fossem corretamente atribuídas entre os seus objetos de software, a fim que o mesmo contemplasse alta coesão e baixo acoplamento visando uma padronização e melhor organização dos seus módulos e componentes. Além claro, de tornar o projeto mais apto a futuras mudanças, per permitindo a melhor compreensão para os desenvolvedores que serão responsáveis pela manutenabilidade do software.

4. Caso de Uso

Um caso de uso corresponde a uma técnica que possui como finalidade auxiliar a documentar os requisitos funcionais de software, de maneira clara e compreensível, garantindo que as informações sejam acessíveis tanto aos stakeholders como aos projetistas do sistema. Para alcançar esse objetivo, ele faz uso de uma linguagem natural, tal como o português ou inglês, mas estrutura a descrição como uma sequência de passos, o que o torna próximo da linguagem computacional. Eles capturam o comportamento do sistema frente à interação com o usuário sob um conjunto de circunstâncias específicas. Além disso, um caso de uso normalmente apresenta um conjunto de atores, estes que representam diferentes pessoas (ou dispositivos) que utilizam os serviços fornecidos pelo sistema para atingir metas particulares.

Os atores desempenham papéis fundamentais ao interagir com o sistema, pois representam entidades externas responsáveis por iniciar ou participar de fluxos de execução nos casos de uso. Cada ator possui objetivos específicos que refletem as funcionalidades desejadas ou necessárias, e sua atuação pode variar conforme o contexto (SOMMERVILLE, 2018). Abaixo são apresentadas as descrições textuais referentes aos casos de uso devolução de livro e emprestar livro. Em seguida mostrado o diagrama de caso de uso referente a devolução de um título além da visão do bibliotecário sobre o sistema.

|  |  |
| --- | --- |
| **Caso de uso:** Devolução de Livro | |
| **Ator:** Usuário da biblioteca | |
| **Pré-Condição:** O livro foi emprestado pelo usuário. A data de devolução foi registrada. | |
| **Fluxo Principal** | **Fluxos Alternativos** |
| 1. O usuário retorna à biblioteca  2. O bibliotecário registra a data de devolução do livro no sistema.  3. O sistema verifica a data de devolução e consta que não houve atraso.  4. O sistema calcula o valor total a ser pago pelo usuário.  5. O usuário não realiza pagamento algum em vista que não há multas.  6. O bibliotecário informa que a devolução foi concluída e não há mais pendencias. | 3.1 O sistema identifica que há atrasos na devolução do livro.    3.1.a. O sistema calcula a multa com base na quantidade de dias de atraso.  3.1.b. O sistema atualiza o valor total a ser pago pelo usuário incluindo a multa calculada. |
| 5.1. O usuário possui multas em vista de devoluções atrasadas de livros emprestados.  5.1.a. O usuário realiza o pagamento da multa de acordo o número de dias em atraso.  5.1.b. Retorna-se ao fluxo principal no passo 6. |
| **Pós-Condição:** O livro é marcado como devolvido no sistema. O usuário não possui mais pendências. | |

**Quadro 1** - Fluxo Principal e Alternativo de Devolução de Livro.

|  |  |
| --- | --- |
| **Caso de uso:** Emprestar Livro | |
| **Ator:** Bibliotecário. | |
| **Fluxo Principal** | **Fluxos Alternativos** |
| 1. O bibliotecário inicia o processo de empréstimo.  2. O sistema solicita ao bibliotecário o número de livros a serem emprestados.  3. O bibliotecário insere o número de livros.  4. O sistema calcula o valor total a ser pago pelo usuário.  4.1. O sistema solicita ao bibliotecário o código do livro.  4.2. O bibliotecário insere o código do livro.  4.3. O sistema registra o livro emprestado.  5. O sistema calcula a data prevista de devolução com base nos livros emprestados.  6. O sistema exibe a data prevista de devolução.  7. O processo de empréstimo é concluído. | 3.1. Se não houver livros disponíveis para empréstimo:  3.1.a. O sistema exibe uma mensagem informando que não há livros disponíveis.  3.1.b. O processo de empréstimo é encerrado. |

**Quadro 2** - Fluxo Principal e Alternativo de Emprestar Livro.

5. Padrões de Projeto

De acordo com Gamma *et al* (2000), Padrões de Projeto se tratam de soluções reutilizáveis para problemas específicos e recorrentes encontrados ao se desenvolver um software orientado a objetos. Eles possuem um nome único que referência um problema, sua solução e as consequências advindo dela, auxiliando na comunicação e na difusão de ideias e conceitos a outros desenvolvedores.

O problema descreve a problemática encontrada e seu contexto servindo de referencial para aplicação do padrão. A solução por outro lado descreve os elementos que compõem o padrão de projeto, seus relacionamentos, responsabilidades e colaborações em uma linguagem abstrata de forma que não é restrita a uma determinada implementação ou linguagem utilizada. Por fim, as consequências listam as vantagens e desvantagens *(trade-offs)* da aplicação do padrão o que auxilia no momento de avaliar alternativas em potencial.

5.1. Model/View/Controller (MVC)

Para modelagem das classes do sistema foi utilizado o padrão MVC (Model/View/Controller) (KRASNER et al, 1988), que tem por característica principal separar a interface com o usuário – as chamadas GUI’s (Graphical UserInterface) – dos dados, propriamente ditos, ou seja, do processamento das informações. Na camada Model são representados os dados. O controle (Controller) responde às interações do usuário com a camada View, que é a interface gráfica. Essa separação de responsabilidades em três componentes principais favorece a modularidade facilitando o desenvolvimento e a manutenção do sistema. MVC é classificado por Gamma et al (2000) como sendo um padrão estrutural.

5.2. Data Access Object (DAO)

Na camada de dados (Model) foi utilizado o padrão DAO (*Data Access Object*), que consiste em permitir o acesso a todo tipo de informação persistente, seja ela em arquivo ou banco de dados, ou seja, é utilizado para obter e armazenar dados e acessos a partir da origem dos dados. Desta forma, um componente de negócio (Controller) fica exposto apenas à interface do DAO, que esconde toda a complexidade relativa à interação com a fonte de dados sendo utilizada, seja ela arquivo, banco de dados etc.

5.3. Padrão em Camadas

No padrão de projeto em camadas, a aplicação é dividida em camadas lógicas, cada uma responsável por um aspecto específico do sistema, promovendo separação de responsabilidades e facilitando a manutenção e escalabilidade. A camada de apresentação (View) é responsável por interagir com o usuário, exibindo informações e capturando entradas. Esta camada se comunica com a camada de controle (Controller), que atua como intermediária, coordenando as interações entre a interface e a lógica de negócios.

A camada de negócios (Model) encapsula as regras e processos do sistema, garantindo que a lógica principal esteja isolada e possa ser reutilizada em diferentes contextos. Por fim, a camada de dados gerencia o acesso às fontes persistentes, como bancos de dados ou arquivos. Ao organizar o sistema desta forma, cada camada conhece apenas a imediatamente inferior, reduzindo o acoplamento entre os componentes e permitindo que mudanças em uma camada tenham impacto mínimo sobre as demais.

X. Referências Bibliograficas

DO AMARAL, R. M.; ZAFALON, Z. R. **Implementação de Sistemas Automatizados em Biblioteca: Contribuição para a Operacionalização da Gestão**. [s.l: s.n.], 2009.

GAMMA, E. et al. **Padrões de Projetos: Soluções Reutilizáveis de Software Orientados a Objetos**. [s.l.] Bookman, 2000.

KRASNER, G. E.; POPE, S. T. A cookbook for using the model view controller user interfaceparadigm in Smalltalk-80. Journal of Object-Oriented Programming. p. 26–49, 1988.

LARMAN, C. **Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development**. 3. ed. Filadélfia, PA, USA: Prentice Hall, 2005.

LIMA, G. A. B. DE O.; PINTO, L. P.; LAIA, M. M. DE. Tecnologia da informação: impactos na sociedade. **Informação & informação**, v. 7, n. 2, p. 75, 2002.

WHITE, H. Fazemos, fazemos, fazemos e não sabemos porque as práticas de catalogação clamam por uma reavaliação. **Revista da Escola de Biblioteconomia da Universidade Federal de Minas Gerais (UFMG)**, p. 257–264, 1993.

WIRFS-BROCK, R.; MCKEAN, A. **Object design: Roles, responsibilities, and collaborations**. Boston, MA, USA: Addison Wesley, 2002.