INSTITUTO SUPERIOR POLITÉCNICO DE TECNOLOGIAS E CIÊNCIAS

DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIAS

ENGENHARIA DE SOFTWARE II

RELATÓRIO DO LABORATÓRIO 7

TRABALHO 2

DESENHO E IMPLEMENTAÇÃO DE SOFTWARE

INTEGRANTES DO GRUPO	
JOSE N'DONGE – 20200689 JÚLIA CAMANA – 20201175 RUI YURI J. MALEMBA – 20201580	
CURSO: ENGENHARIA INF6 **TURMA:** M1	**DOCENTE:** JUDSON PAIVA
Data de Realização do Projecto: 16 de Junho de 2023	

Índice

A. Desenho Orientado a Objetos com UML:

1. Quais são os conceitos fundamentais da programação orientada a objetos e como eles se relacionam com o desenho de software?

Resposta:

Os conceitos fundamentais são:

- ✓ **Abstração:** A abstração é a capacidade de representar características essenciais de um objeto do mundo real no código. Isso permite que você modele objetos relevantes para o seu sistema, capturando seus atributos e comportamentos relevantes, enquanto oculta detalhes irrelevantes. A abstração ajuda a simplificar o design do software, tornando-o mais compreensível e manutenível.

- ✓ **Encapsulamento:** O encapsulamento envolve o agrupamento de dados e métodos relacionados em um único objeto. Os dados são protegidos dentro do objeto e podem ser acessados somente por meio de métodos específicos, conhecidos como métodos de acesso. O encapsulamento ajuda a ocultar a implementação interna do objeto, fornecendo uma interface consistente e controlada para interagir com o objeto. Isso promove a modularidade e a reutilização do código.

- ✓ **Herança:** A herança permite que uma classe herde características de outra classe. Uma classe derivada (subclasse) pode herdar atributos e métodos de uma classe base (superclasse) e, em seguida, adicionar ou modificar comportamentos específicos. A herança promove a

reutilização de código e ajuda a estabelecer relações hierárquicas entre classes.

- ✓ **Polimorfismo:** O polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme. Isso significa que um objeto pode ser referenciado por meio de uma classe base, mas executar o comportamento específico da classe derivada na qual ele pertence. O polimorfismo ajuda a escrever código mais genérico e flexível, facilitando a extensibilidade e a manutenção do software.

Esses conceitos da POO estão intimamente relacionados com o desenho de software. Eles ajudam a dividir o sistema em componentes modulares e coesos, permitindo uma melhor organização, manutenção e reutilização do código. A abstração e o encapsulamento ajudam a ocultar a complexidade e fornecer interfaces claras entre os módulos. A herança e o polimorfismo permitem criar hierarquias e relacionamentos entre os objetos, facilitando a extensibilidade e a adaptação do software às mudanças nos requisitos. Ao aplicar esses conceitos, é possível desenvolver um software mais robusto, flexível e escalável.

2. Como podem ser utilizados os diagramas UML para representar a estrutura e o comportamento de um software?

Resposta:

Os diagramas UML (Unified Modeling Language) são amplamente utilizados para representar a estrutura e o comportamento de um software de forma visual e padronizada. Eles fornecem uma linguagem comum para comunicar conceitos e ideias entre desenvolvedores, analistas de sistemas e outros envolvidos no processo de desenvolvimento de software.

Abaixo estão alguns dos principais diagramas UML que podem ser usados para representar a estrutura e o comportamento de um software:

- ✓ **Diagrama de Classes:** Esse diagrama representa a estrutura estática do sistema, mostrando as classes, seus atributos, métodos e os relacionamentos entre elas. É útil para visualizar a organização das classes e a hierarquia de herança.

- ✓ **Diagrama de Objetos:** Esse diagrama representa uma instância específica de um objeto em um determinado momento. Ele mostra os objetos, seus atributos e os relacionamentos entre eles. É útil para entender como os objetos interagem durante a execução do sistema.

- ✓ **Diagrama de Componentes:** Esse diagrama representa os componentes do sistema e as dependências entre eles. Ele mostra como o software é dividido em partes independentes e como essas partes se relacionam para formar o sistema.

- ✓ **Diagrama de Pacotes:** Esse diagrama organiza os elementos do sistema em grupos lógicos chamados de pacotes. Ele mostra a estrutura de pacotes e as dependências entre eles. É útil para visualizar a divisão lógica e modularização do sistema.

- ✓ **Diagrama de Sequência:** Esse diagrama representa a interação entre objetos ao longo do tempo. Ele mostra a ordem em que as mensagens são trocadas entre os objetos durante uma determinada sequência de eventos. É útil para modelar o comportamento dinâmico do sistema.

- ✓ **Diagrama de Atividades:** Esse diagrama descreve o fluxo de atividades dentro de um processo ou função. Ele mostra as etapas sequenciais, as decisões, as ramificações e os loops. É útil para modelar o comportamento de um algoritmo, uma operação ou um caso de uso.

3. Quais são as diferenças entre os principais diagramas UML, como diagrama de classes, diagrama de sequência e diagrama de atividades?

   Resposta:

   Resumidamente, as principais diferenças entre os diagramas UML mais utilizados são:

   Diagrama de Classes:

   - ✓ Finalidade: Representa a estrutura estática do sistema, mostrando as classes, seus atributos, métodos e relacionamentos.
   - ✓ Enfoque: Concentra-se na modelagem dos elementos estruturais do sistema, como classes e suas relações.

   Diagrama de Sequência:

   - ✓ Finalidade: Representa a interação entre objetos ao longo do tempo, mostrando a ordem das mensagens trocadas entre eles.
   - ✓ Enfoque: Concentra-se na modelagem do comportamento dinâmico do sistema, mostrando a colaboração entre objetos.

   Diagrama de Atividades:

- ✓ Finalidade: Descreve o fluxo de atividades em um processo ou função, mostrando as etapas sequenciais, decisões e ramificações.
- ✓ Enfoque: Concentra-se na modelagem do comportamento do sistema como um conjunto de atividades e fluxos de controle.

Em resumo, o diagrama de classes enfoca a estrutura estática do sistema, o diagrama de sequência destaca a interação temporal entre os objetos e o diagrama de atividades modela o fluxo de atividades e controle do sistema. Cada um desses diagramas tem seu propósito e fornece uma visão específica do software, permitindo uma compreensão mais completa do sistema em diferentes níveis de abstração.

B. Desenvolvimento de Interfaces e Funcionalidades:

1. Quais são as tecnologias e frameworks mais utilizados para o desenvolvimento de interfaces gráficas de usuário?

Resposta:

As tecnologias e frameworks mais utilizados para o desenvolvimento de interfaces gráficas de usuário podem variar dependendo da plataforma e do tipo de aplicação. Alguns dos mais populares são:

- ✓ Para desenvolvimento de interfaces web: HTML, CSS, JavaScript, frameworks como React, Angular e Vue.js.
- ✓ Para desenvolvimento de interfaces mobile (Android): Java, Kotlin, frameworks como Android SDK, Flutter, React Native.

- ✓ Para desenvolvimento de interfaces desktop (Windows, macOS, Linux): C#, Java, frameworks como Windows Presentation Foundation (WPF), JavaFX, Electron.

2. Quais são os princípios de design de interfaces que devem ser considerados ao desenvolver uma aplicação?

   Resposta:

   Ao desenvolver uma aplicação com interface gráfica, é importante considerar os seguintes princípios de design de interfaces:

   - ✓ Usabilidade: A interface deve ser fácil de usar, intuitiva e eficiente para o usuário. Deve ser levado em conta o fluxo de trabalho do usuário e a organização dos elementos de interface para facilitar a interação.
   - ✓ Consistência: Os elementos de interface devem seguir um padrão consistente em toda a aplicação, desde o layout até a disposição dos controles e a forma como as ações são realizadas. Isso ajuda os usuários a se familiarizarem com a aplicação e a reduzir erros.
   - ✓ Feedback: A interface deve fornecer feedback visual e/ou auditivo para indicar ao usuário o resultado de suas ações. Por exemplo, exibindo mensagens de confirmação, mostrando animações de carregamento ou fornecendo indicadores visuais do estado atual.
   - ✓ Eficiência: A interface deve ser projetada para permitir que o usuário realize suas tarefas de forma eficiente. Isso inclui a minimização de cliques, a utilização de atalhos de teclado, a automação de tarefas repetitivas e a otimização do desempenho da aplicação.
   - ✓ Estética: Uma interface atraente e bem projetada pode melhorar a experiência do usuário. A escolha de cores, tipografia e elementos

visuais deve ser feita com cuidado, levando em consideração a identidade visual da aplicação e a preferência do público-alvo.

3. Como podem ser implementadas funcionalidades de forma eficiente e robusta, levando em conta aspectos como manipulação de eventos e persistência de dados?

   Resposta:

   Para implementar funcionalidades de forma eficiente e robusta, considerando aspectos como manipulação de eventos e persistência de dados, é recomendado seguir boas práticas de desenvolvimento de software. Algumas sugestões incluem:

   - ✓ Utilizar padrões de projeto: Os padrões de projeto oferecem soluções comprovadas para problemas comuns de desenvolvimento de software. Por exemplo, o padrão Observer pode ser utilizado para a manipulação de eventos, enquanto o padrão DAO (Data Access Object) pode ser utilizado para a persistência de dados.
   - ✓ Separar a lógica de negócio da interface: É importante manter a lógica de negócio separada da interface gráfica. Isso facilita a manutenção, o teste e a reutilização de código.
   - ✓ Utilizar bibliotecas e frameworks: O uso de bibliotecas e frameworks pode acelerar o desenvolvimento, fornecer recursos pré-construídos e ajudar a lidar com desafios comuns. Por exemplo, frameworks como React ou Angular possuem recursos para manipulação de eventos e gerenciamento de estado.
   - ✓ Fazer uso de práticas de otimização: Para garantir uma implementação eficiente, é importante considerar a otimização de algoritmos, o uso adequado de estruturas de dados e a minimização

de operações custosas. Realizar testes de desempenho e identificar gargalos também pode ser útil.

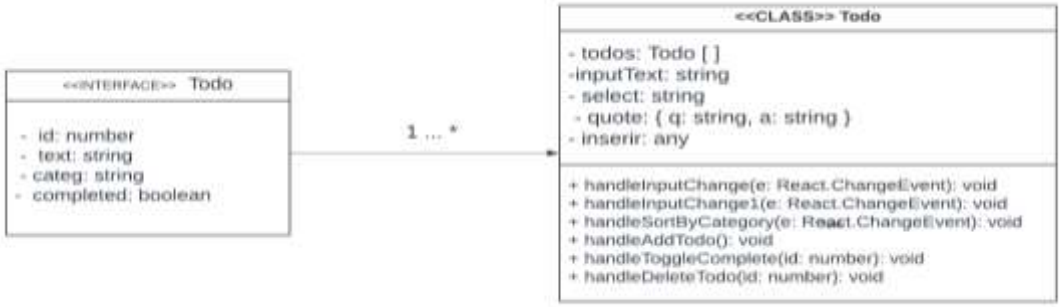
- ✓ Utilizar técnicas de persistência de dados apropriadas: A escolha da tecnologia de persistência de dados depende das necessidades do projeto. Pode-se utilizar bancos de dados relacionais (como MySQL, PostgreSQL) ou não relacionais (como MongoDB, Redis). É importante considerar fatores como escalabilidade, segurança e eficiência das consultas.

C. DESENVOLVIMENTO DO SISTEMA DE GERENCIAMENTO DE TAREFAS

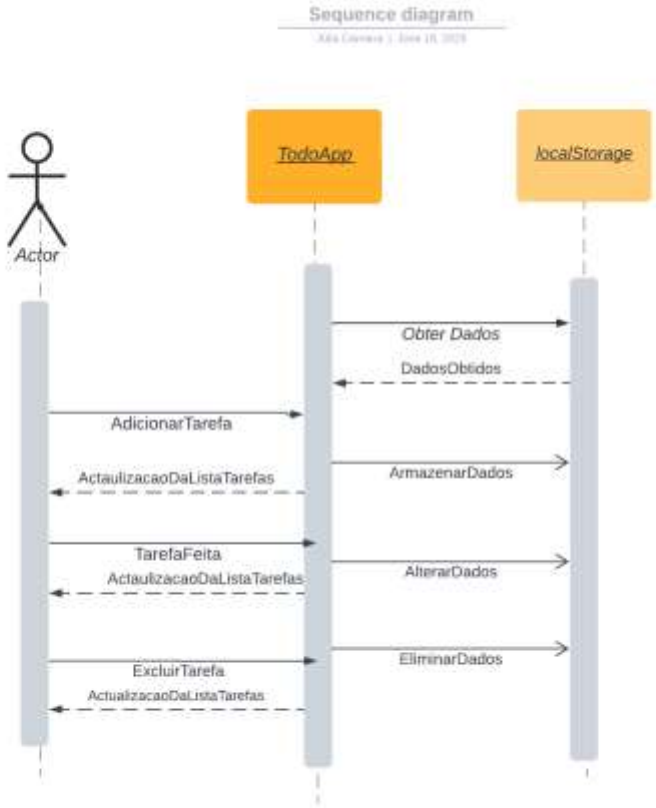
1. Desenho Orientado a Objetos com UML:

- ✓ Análise de requisitos:
  - Inserir tarefa
  - Remover tarefa
  - Marcar como concluída
  - Listar tarefas
  - Listar tarefas por categoria

- ✓ Diagrama de classe



✓ Diagrama de sequência



2. Desenvolvimento de Interfaces e Funcionalidades:

✓ Tecnologia ou framework para o desenvolvimento da interface gráfica de usuário: React js

✓ Implementamos as funcionalidades do sistema, incluindo a criação, exclusão de tarefas, organização em categorias, e a possibilidade de marcar tarefas como concluídas, usando conceitos de react hooks como o useState e o useEffect. Guardamos os dados inseridos, na local storage por opção técnica após uma discussão da eauipe de desenvolvimento.

- ✓ Telas do sistema considerando os princípios de design de interfaces, como usabilidade, consistência visual e feedback ao usuário:

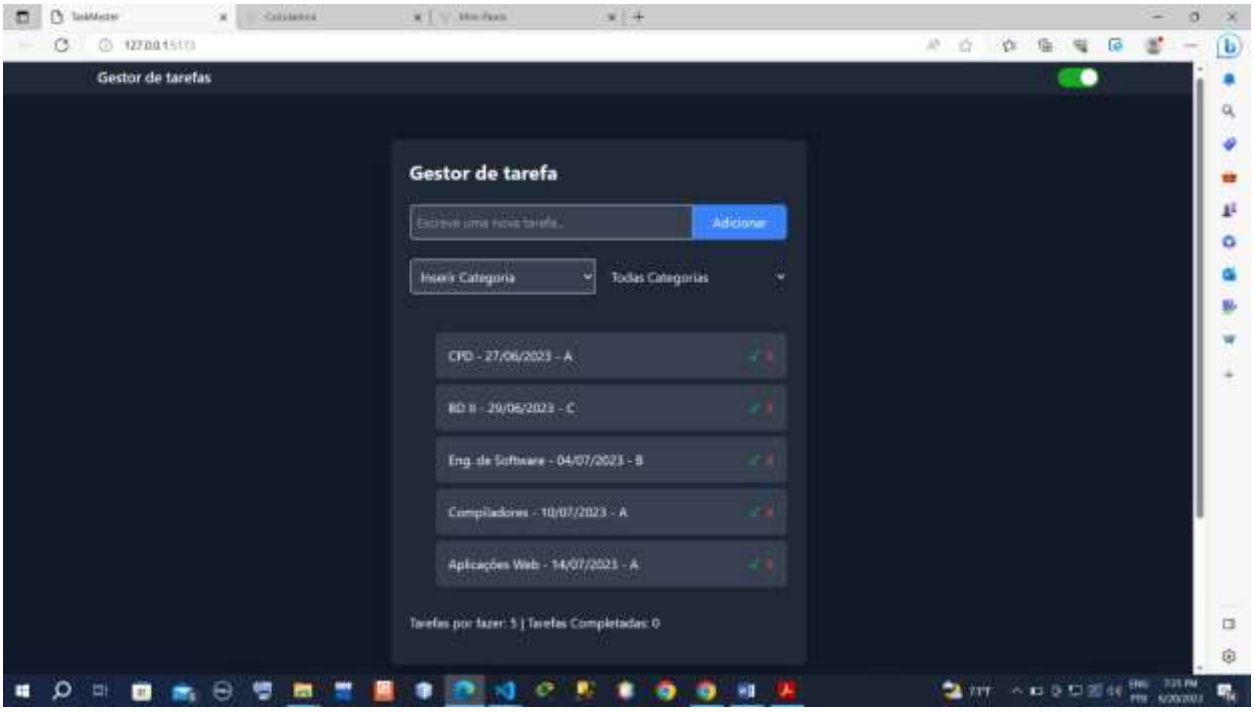


Figura 1- Tela do gestor no modo escuro

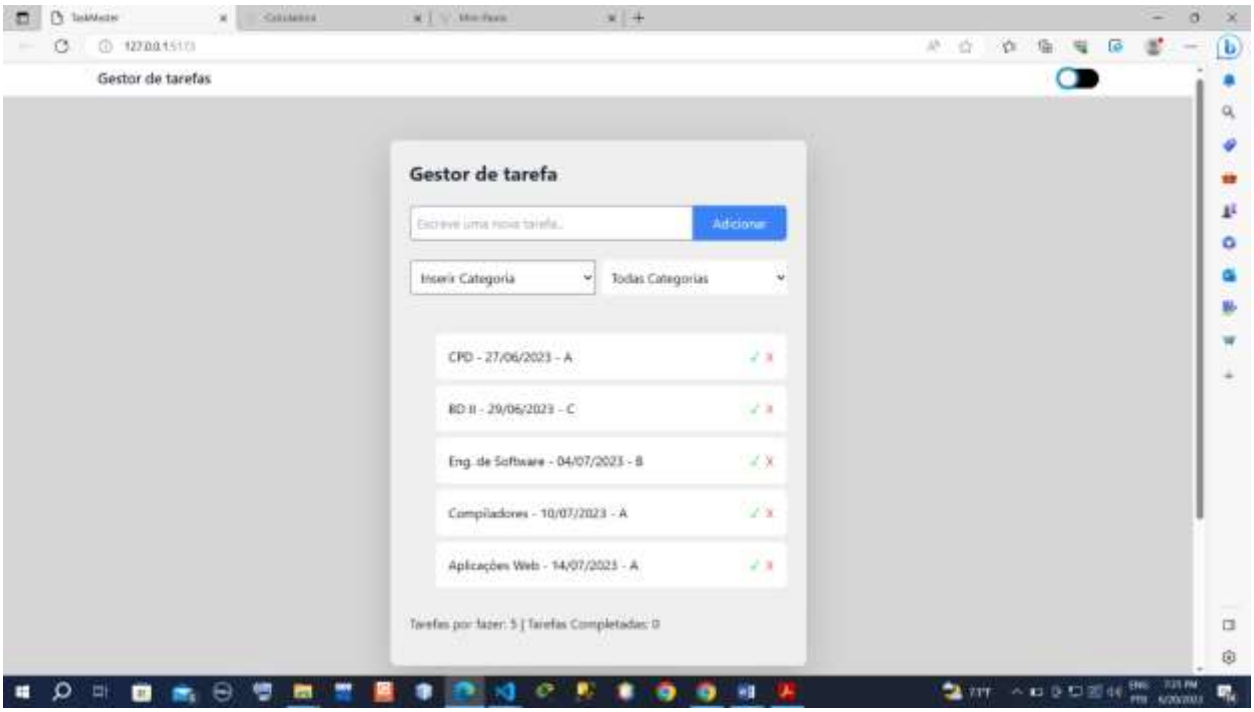


Figura 2 - Gestor no modo claro