INSTITUTO FEDERAL
Minas Gerais
Campus
Ouro Branco

Trabalho De Sistemas de Apoio a Decisão

Aluno: Pedro Arthur Cunha Anício / 0072383

1. Introdução:

O sistema Corporate Financial Control tem como objetivo auxiliar empresas na gestão financeira por meio de um sistema de apoio a decisão completo e eficiente. Ele possibilita o controle de receitas e despesas financeiras, permitindo que o contador de determinada empresa tenha acesso à relatórios detalhados e a visualização gráfica dos dados financeiros.

No âmbito empresarial, a análise de dados financeiros é fundamental para tomadas de decisões assertivas, e esse sistema procura solucionar esses problemas e facilitar a identificação de problemas na área de finanças.

2. Arquitetura do Sistema:

O Corporate Financial Control foi desenvolvido utilizando a arquitetura em camadas, que promove uma organização clara e a separação de responsabilidades no código.

1. Camada de modelo (Model):

    - Apresenta as entidades(Objetos) do sistema, nesse caso, categorias, receitas e despesas;
    - Inclui classes que encapsulam os dados e fornecem estrutura para a interação com o banco de dados.

2. Camada de Acesso aos Dados (Repository):

    - Realiza operações diretas no banco de dados, por meio de SQL queries;
    - Implementada utilizando JDBC para garantir um maio controle e segurança contra ataques, como SQL Injection.

3. Camada de serviço (Service):

    - Aplica as regras de negócio e processa os dados antes de enviá-los para a camada de controle.

4. Camada de Controle (Controller):

    - Gerencia a comunicação entre o backend e frontend;
    - Expõe endpoints RESTful que permitem a interação com as funcionalidades do sistema.

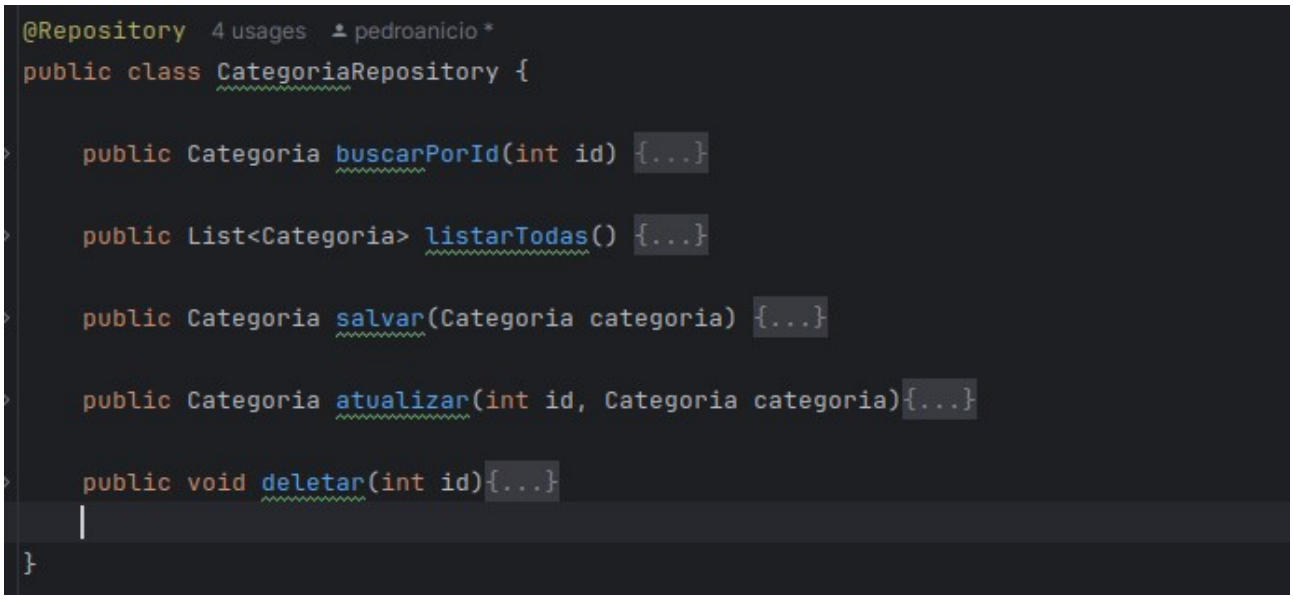
5. Camada de Visualização (FrontEnd):
    - Desenvolvida com React, oferece uma interface gráfica para a visualização detalhada dos dados;
    - Inclui dashboards, gráficos e formulários para inserção, edição e exclusão de dados.

A integração entre essas camadas garante que o sistema seja modular, escalável e de fácil manutenção. Além disso, o uso de boas práticas, como os princípios SOLID, foi aplicado durante o desenvolvimento para aumentar a qualidade do código. O projeto foi desenvolvido utilizando Spring, Java, React e MySQL.

Demonstração da utilização de alguns dos princípios SOLID:

1. **Princípio da Responsabilidade única:**

Definição: Cada classe ou módulo deve ter apenas uma razão para mudar, ou seja, deve possuir uma única responsabilidade.
Exemplo no Código: A classe “CategoriaRepository”, por exemplo, é responsável somente por interagir com a base de dados em relação à entidade “Categoria”. Todas as operações, como buscar, salvar, editar e deletar categorias estão centralizadas em apenas uma classe.



2. **Princípio da Inversão de dependência:**

Definição: Módulos de alto nível não devem depender de módulos de baixo nível; ambos devem depender de abstrações.
Exemplo: A configuração de conexão com o banco de dados utiliza o DatabaseConfig, promovendo o desacoplamento. Em vez de criar conexões diretamente dentro da lógica de repositórios, a conexão é fornecida por uma classe separada, facilitando a troca ou modificação do banco de dados sem impactar o restante do sistema.

```java
@Repository  4 usages  pedroanicio *
public class CategoriaRepository {

    public Categoria buscarPorId(int id) {  5 usages  pedroanicio *
        Categoria categoria = null;
        String sql = "SELECT * FROM categorias WHERE id = ?";

        // Conexão fornecida pelo DatabaseConfig
        try (Connection conn = DatabaseConfig.getConnection();
             PreparedStatement stmt = conn.prepareStatement(sql)) {
```

3. **Princípio de Substituição de Liskov:**

Definição: Objetos de uma classe derivada devem poder substituir objetos da classe base sem alterar o comportamento correto do programa.
Exemplo: Embora isso não seja explícito no código, as operações no repositório seguem um contrato previsível para lidar com entidades. Por exemplo, métodos como salvar, atualizar, e deletar utilizam o mesmo padrão e comportamento esperado para manipular a entidade Categoria. Se futuramente for implementada uma Interface Repository para diferentes entidades, todos os repositórios poderão substituí-la, garantindo compatibilidade e previsibilidade.

```java
public interface Repository<T> {  no usages
    T buscarPorId(int id);  no usages
    List<T> listarTodas();  no usages
    T salvar(T entidade);  no usages
    T atualizar(int id, T entidade);  no usages
    void deletar(int id);  no usages
}
```

3. Funcionalidades do Sistema:

1. **Gestão de receitas e despesas:**

Possibilidade de adicionar, deletar e editar todas as movimentações financeiras. Cada receita e despesa apresenta uma categoria específica, que também pode ser adicionada pelo usuário. Por exemplo, “Aluguel, “Salários”, etc. Cada despesa apesenta um meio de pagamento, que foram definidos como ENUMs(Cartão, dinheiro ou transferência).

2. **Geração de relatórios financeiros:**

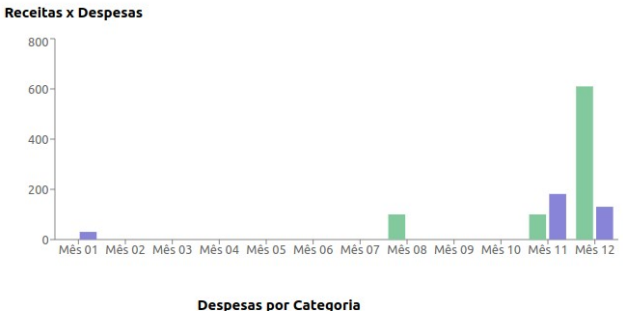
O frontend permite a visualização concreta desses dados por meio de gráficos intuitivos e completos. Também permite a visualização desses dados por meio de tabelas, onde o usuário pode adicionar, deletar e atualizar aqueles dados.

Sistema de Apoio à Decisão - Dashboard

Receita Total
R$ 810,04

Despesa Total
R$ 342,5

Saldo Atual
R$ 467,54



Despesas por Categoria

Gráfico pizza:

Relatórios:

Relatórios Detalhados

Receitas

ID	Descrição	Valor	Data	Categoria
5	Venda de produto	R$ 300	2024-12-03	Investimentos
6	Venda de produto	R$ 100	2024-12-03	Investimentos
7	Venda de produto	R$ 100,01	2024-12-03	Investimentos
8	Venda de produto	R$ 100,01	2024-11-03	Investimentos
9	Venda de produto	R$ 100,01	2024-08-03	Investimentos
10	Venda de produto	R$ 100,01	2024-12-03	Investimentos
11	teste	R$ 10	2024-12-09	Investimentos

Despesas

ID	Descrição	Valor	Data	Categoria
9	Compra de materiais de escritório	R$ 140,5	2024-11-06	Investimentos

4. Segurança e Integridade:

1. **Proteção contra SQL Injection:**

- O sistema utiliza “PreparedStatement” em todas as interações com o banco de dados, evitando que entradas maliciosas sejam interpretadas como comandos SQL.

2. **Tratamento de Erros e Exceções:**

- Manipulação de exceções para ocultar mensagens detalhadas de erros do banco de dados.
- Logs seguros para monitoramento e auditoria, sem expor dados críticos.

3. **Controle de Acesso e Autenticação:**

- Uso de autenticação para validar usuários antes de acessar o sistema.
- Restrições de permissão para certas operações, como exclusões ou modificações.

4. **Validação de Dados no Backend:**

- Validação de entradas do usuário no backend antes de salvar ou processar informações.
- Exemplo: verificar se campos obrigatórios estão preenchidos, se os valores são positivos (para receitas/despesas) ou se respeitam limites esperados.

5. **Estrutura Modular e Segura:**

- Seguindo os princípios SOLID, cada classe tem uma responsabilidade única, tornando o sistema mais fácil de manter e reduzindo o risco de erros de segurança.

5. Conclusão:

O sistema de administração financeira empresarial criado une solidez, eficácia e segurança para satisfazer as demandas de gerenciamento de informações financeiras de maneira segura. Através de uma estrutura em camadas que adere aos princípios SOLID, o sistema assegura uma manutenção descomplicada, expansibilidade e a adoção de boas práticas de programação.

Cada fase do desenvolvimento foi dedicada à segurança, utilizando métodos como “PreparedStatement” para prevenir Injection de SQL, validação de dados, correção de falhas e planejamento para futuras implementações de criptografia e autenticação sólida. Tais ações garantem a integridade e a confidencialidade das informações.

As funcionalidades implementadas, aliadas à estrutura modular, permitem uma experiência de uso prática e eficiente, suportada por um backend sólido e gráficos que facilitam a visualização e análise de dados financeiros. O sistema se apresenta como uma solução confiável para empresas

que buscam organizar e tomar decisões financeiras de forma estratégica e segura.