Sistema de Gerenciamento de Congressos Empresariais

Trabalho apresentado à Disciplina de Programação Modular

Trabalho de Prática Investigativa

Gabriela Alvarenga Cardoso

Bernardo Souza Alvim

Joaquim Guilherme de Carvalho Vilela Silva

Marcos Alberto Ferreira Pinto

Vitor Costa Vianna

Orientador: Hugo Bastos de Paula
Orientador do Trabalho de Prática Investigativa, Brasil– hugo@pucminas.br

Pontifícia Universidade Católica de Minas Gerais

Belo Horizonte, Novembro de 2024

Resumo

Este trabalho apresenta o desenvolvimento de um sistema de gerenciamento de congressos empresariais, integrando dados de profissionais, fornecedores, atividades e equipes de apoio para otimizar a organização e execução de eventos. O projeto utiliza Programação Orientada a Objetos com a criação de pelo menos cinco classes inter-relacionadas, especificações detalhadas e arquitetura modular. Além disso, inclui a geração de código com LLMs, abrangendo a elaboração de prompts, refinamento iterativo, avaliação da qualidade do código gerado e análise crítica das limitações e melhorias dos modelos, garantindo aderência a princípios da POO e qualidade do software. modular que permite futuras expansões do sistema.

Palavras-chave:Gerenciamento de Congressos, Eventos Corporativos, Programação Orientada a Objetos, Geração de Código, Qualidade de Software.

1 ESPECIFICAÇÃO DO PROJETO

1.1 Sistema de Gerenciamento de Congressos Empresariais

1.2 Introdução

1.2.1 Contextualização do Problema

Empresas responsáveis por organizar congressos, conferências e eventos corporativos enfrentam desafios significativos na gestão integrada de diversos aspectos, como profissionais, fornecedores, cadastro de atividades e equipes de apoio. A ausência de um sistema centralizado pode levar a falhas logísticas, inconsistências nos dados e insatisfação dos clientes, comprometendo a qualidade dos eventos e a reputação das empresas. Este projeto busca solucionar esses problemas por meio de uma plataforma robusta que centralize e otimize a gestão de congressos empresariais, garantindo eficiência, organização e satisfação dos clientes.

1.2.2 Objetivo do Projeto

Desenvolver um sistema de gerenciamento de congressos empresariais que permita às empresas organizadoras administrar, de forma integrada e flexível, todas as etapas de planejamento e execução de eventos corporativos. A plataforma oferecerá funcionalidades completas, como o cadastro de profissionais, fornecedores, atividades e gerenciamento financeiro. Além disso, permitirá o cadastro de pacotes de serviço, incluindo a definição e associação de itens de serviço específicos.

1.3 Requisitos do Software

1.3.1 Requisitos Funcionais

- **Cadastro de Congresso**: Cadastrar informações sobre o congresso, incluindo setor, valor e atividades relacionadas.
- **Cadastro de Atividades**: Registrar atividades como Painel de Discussão, Networking, Palestra e Workshop, incluindo tipo, data, local e participantes.
- **Gerenciamento de Atribuições**: Controlar atribuições de profissionais com datas e cargas horárias.
- **Cadastro de Profissionais**: Registrar profissionais com nome, CPF, papel e tempo alocado.
- **Cadastro de Fornecedores**: Cadastrar fornecedores com nome, CNPJ, serviço e endereço.
- **Pacotes de Serviço**: Cadastrar pacotes e gerenciar itens associados.
- **Cálculo de Custos**: Calcular automaticamente o custo total dos pacotes.
- **Filtragem e Ordenação de Profissionais**: Filtrar por papel e ordenar por tempo alocado.
- **Filtragem de Atividades**: Filtrar atividades por data.
- **Ordenação de Itens de Serviço**: Ordenar itens de serviço por custo.

1.3.2 Requisitos Não Funcionais

- **Escalabilidade**: Permitir adição de novas funcionalidades de forma simples.
- **Manutenibilidade**: Facilitar manutenções e evolução do sistema.
- **Usabilidade**: Interface amigável para cadastro e consultas.

2 MODELAGEM DO SISTEMA: GESTÃO DE EVENTOS CORPORATIVOS

2.1 Diagrama de Classes

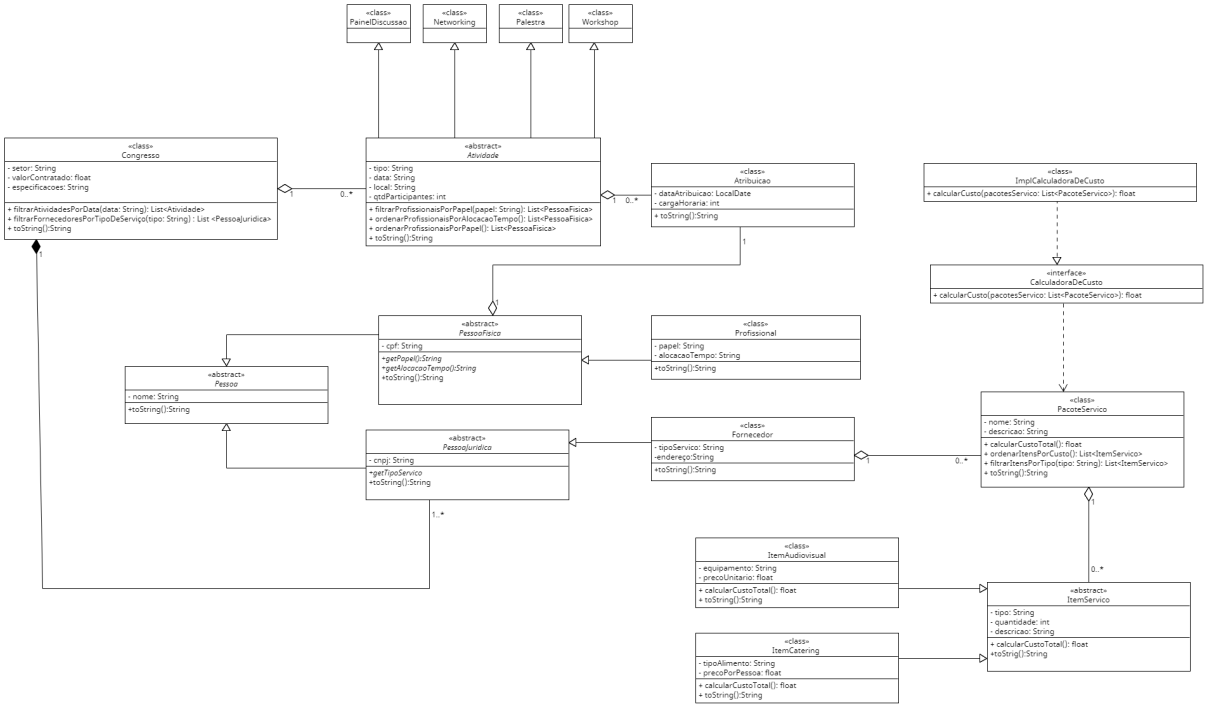

Figura 1 – Outro Diagrama de Classes ou Detalhamento Adicional

2.1.1 Abordagem de Modelagem

Os requisitos do software já fornecem uma visão clara das inter-relações entre as classes do sistema. Cada funcionalidade descrita, como o gerenciamento de atividades, atribuições e cálculos de custos, reflete diretamente a interação esperada entre entidades como *Congresso*, *Atividade*, *Pessoa* e *Pacote de Serviço*. Por isso, consideramos que criar uma seção específica apenas para descrever as inter-relações entre classes seria redundante, uma vez que essas conexões estão implícitas nos próprios requisitos funcionais.

O diagrama de classes apresentado reflete a abordagem que nosso grupo adotou para implementar o projeto. Desenvolvemos a modelagem com base nos princípios SOLID, garantindo que o sistema fosse modular, coeso e de fácil manutenção.

3 METODOLOGIA DE DESENVOLVIMENTO

3.1 Abordagem de Programação

A metodologia de desenvolvimento adotada será a abordagem **Agile**, com sprints curtos para entregas frequentes e feedback contínuo.

3.2 Ferramentas e Tecnologias

- **VSCode**: IDE para desenvolvimento.
- **GitHub**: Controle de versão.
- **Java**: Linguagem principal.
- **Maven**: Gerenciador de dependências.
- **JFrame**: Biblioteca para a interface gráfica.

4 GERAÇÃO DE CÓDIGO COM LLMS

4.1 Escolha do Modelo

- **Modelo Selecionado:** Gemini IA
- **Tamanho e Arquitetura do Modelo:** Modelo otimizado para sugestões contextuais em uma ampla gama de tarefas de desenvolvimento.
- **Capacidade de Geração de Código:** Suporte para autocompletar, criação de funções complexas e depuração em várias linguagens (Python, JavaScript, Java, C++), com aumento de produtividade em até 40%.
- **Suporte para Programação Orientada a Objetos (OOP):** Habilidade em estruturar classes e métodos com boas práticas de design, incluindo herança e polimorfismo.
- **API Disponível:** API bem documentada para integração com editores de código e IDEs, compatível com diversas linguagens e frameworks populares.
- **Relevância do Google:** O Google é líder no mercado de IA e tecnologia, e o modelo Gemini IA reflete sua capacidade de oferecer soluções inovadoras que aumentam a produtividade no desenvolvimento de software.

4.2 Engenharia de Prompts

A engenharia de prompts adotará a abordagem de **Zero-shot Prompting**, onde o modelo gera respostas adequadas sem exemplos específicos, utilizando seu conhecimento prévio para entender a tarefa com base na descrição fornecida.

- **Descrição do Problema:** O modelo entenderá o problema com base na descrição clara e nos requisitos funcionais.
- **Especificação das Classes e seus Atributos:** Serão listadas as classes e seus atributos, com tipos de dados e restrições, permitindo que o modelo gere o código adequado.
- **Relações entre as Classes:** As relações entre as classes, como herança e composição, serão explicadas para garantir a estrutura correta da aplicação.
- **Métodos e Funcionalidades de Cada Classe:** Serão especificados os métodos e funcionalidades de cada classe para que o modelo compreenda as interações e gere o código correspondente.

4.3 Iteração e Refinamento

O processo de iteração e refinamento assegurou que o código atendesse aos requisitos do sistema de gerenciamento de congressos empresariais. A cada ciclo, o feedback foi analisado para ajustar os prompts e melhorar a saída.

- **Processo de Iteração:** O feedback foi utilizado para identificar melhorias e ajustar a implementação conforme os requisitos.
- **Refinamento dos Prompts:** Os prompts foram ajustados para incluir as regras de negócio, validações nos setters e a criação de um construtor refinado.

5 AVALIAÇÃO DO CÓDIGO

5.1 Adesão aos Princípios da POO

O código gerado foi avaliado segundo os princípios da Programação Orientada a Objetos (POO), garantindo o uso de boas práticas como:

- **Princípios Avaliados**: Encapsulamento, herança, polimorfismo, coesão e acoplamento.

No geral, o modelo generativo escolhido apresentou bom desempenho no contexto da Programação Orientada a Objetos, estruturando as entidades de forma coerente e atendendo aos requisitos básicos dos princípios avaliados.

5.2 Qualidade do Código

Critérios específicos foram utilizados para avaliar a qualidade do código gerado.

- **Critérios de Avaliação**: Foram considerados aspectos como legibilidade, eficiência e facilidade de manutenção do código.

A legibilidade dos códigos gerados é satisfatória, com boa organização estrutural e consistência no uso de identação e nomenclatura de variáveis. Essa clareza facilita a compreensão, mesmo para desenvolvedores com menor experiência no domínio.

A eficiência do código foi avaliada pela capacidade de executar as tarefas propostas de maneira otimizada, evitando operações redundantes e complexidades desnecessárias.

A modularidade apresentada contribui para a facilidade de manutenção, permitindo que alterações ou expansões sejam implementadas sem comprometer outras partes do sistema.

5.3 Comparação com Código Humano

Alterações no código gerado foram propostas para aprimorar sua aderência aos princípios da POO e aos padrões comuns de código humano.

- **Propostas de Alterações**: Sugestões para adequar o código aos princípios da POO e aos padrões de design, como os princípios SOLID.

  Embora o modelo generativo tenha se mostrado eficiente em estruturar códigos baseados em instruções específicas, o aumento da complexidade dos prompts evidenciou limitações.

  Quando instruído a gerar código com alta aderência aos princípios SOLID, o modelo frequentemente dissociou parte das instruções mais complexas. Isso demonstra que, embora seja eficaz para tarefas diretas, há desafios em equilibrar múltiplos requisitos de design em códigos mais sofisticados.

6 ANÁLISE DOS RESULTADOS

6.1 Identificação de Limitações

Limitações específicas dos LLMs na geração de código foram identificadas e classificadas em pontos principais:

- **Complexidade do Código**: Os LLMs enfrentam dificuldades ao lidar com requisitos complexos que envolvem múltiplos princípios de design ou integração de funcionalidades interdependentes.

- **Ambiguidades nas Instruções**: Instruções pouco claras ou abertas a múltiplas interpretações resultam em códigos inconsistentes ou desalinhados com as expectativas do desenvolvedor.
- **Respeito a Padrões de Design**: Apesar de seguir diretrizes básicas, os LLMs frequentemente não aderem completamente a padrões avançados, como os princípios SOLID.
- **Falta de Contexto Avançado**: Em projetos que exigem conhecimento detalhado de domínios específicos, os LLMs podem gerar soluções genéricas ou não otimizadas.

6.2 Propostas de Melhoria

Com base nas limitações observadas, foram sugeridas diversas formas de utilizar a IA existente de maneira mais eficiente no contexto de geração de código:

- **Divisão de Problemas Complexos em Subtarefas**: Estruturar os prompts em etapas menores e mais específicas, permitindo que a IA gere partes do código que possam ser integradas posteriormente pelo desenvolvedor.
- **Refinamento Progressivo por Iterações**: Trabalhar iterativamente com a IA, fornecendo feedback em ciclos curtos para ajustar e refinar o código gerado em alinhamento com os padrões esperados.
- **Fornecimento de Exemplos Claros no Prompt**: Incluir no prompt exemplos de código similares ao esperado, reduzindo ambiguidades e guiando a IA para soluções mais alinhadas com os padrões desejados.
- **Validação Automatizada do Código**: Utilizar ferramentas de linting e análise estática para avaliar e corrigir automaticamente problemas no código gerado, garantindo melhor aderência às boas práticas.
- **Exploração de Geração de Documentação**: Aproveitar a capacidade da IA para gerar documentação explicativa para o código produzido, auxiliando na compreensão das decisões implementadas.
- **Contextualização Adicional com Padrões de Design**: Incluir descrições detalhadas de padrões de design e princípios de POO nos prompts, para alinhar o código gerado com as melhores práticas.
- **Utilização da IA como Ferramenta de Brainstorming**: Em vez de depender inteiramente do código gerado, usar a IA para explorar soluções alternativas ou gerar insights iniciais que possam ser refinados manualmente.

- **Comparação Automática com Padrões de Código**: Usar ferramentas para comparar o código gerado com padrões de código humanos, identificando rapidamente discrepâncias e áreas para ajuste.

7 CONCLUSÃO

7.1 Resumo dos Resultados

O uso de LLMs para a geração de código demonstrou ser promissor, mas também expôs limitações relevantes. Entre os principais resultados, destacam-se:

- A capacidade dos modelos de gerar códigos que aderem, em boa medida, aos princípios básicos da Programação Orientada a Objetos (POO), como encapsulamento, herança e polimorfismo.
- Dificuldades em atender de forma consistente aos padrões avançados de design, como os princípios SOLID, especialmente em prompts com maior complexidade ou ambiguidade.
- Alta legibilidade e clareza do código gerado, ainda que ajustes fossem necessários para garantir eficiência e aderência total às boas práticas de engenharia de software.
- A relevância da abordagem iterativa, onde o desenvolvedor age como mediador, refinando o código gerado por meio de feedback progressivo e complementando as lacunas identificadas.

7.2 Considerações Finais

A geração de código por LLMs tem um potencial significativo para revolucionar o desenvolvimento de software, especialmente em tarefas repetitivas ou baseadas em padrões bem definidos. No entanto, os resultados indicam que, para alcançar a máxima utilidade, é necessário:

- Estruturar prompts com clareza, dividindo problemas complexos em subtarefas e fornecendo exemplos específicos que guiem o modelo em direção às soluções esperadas.
- Utilizar as IAs como ferramentas de brainstorming e validação, aproveitando sua capacidade de explorar diferentes abordagens e identificar possíveis melhorias no código.
- Complementar a geração automática com ferramentas de linting e validação de padrões para garantir que o código final seja eficiente, escalável e aderente às boas práticas.

- Contextualizar os LLMs com informações mais ricas sobre o domínio e o problema, permitindo-lhes gerar soluções mais alinhadas com as necessidades do projeto.

  Embora os modelos generativos mostrem limitações em cenários complexos, o estudo reforça seu valor como assistentes de programação. A combinação de suas capacidades com a supervisão humana pode impulsionar o desenvolvimento de software, facilitando a criação de códigos mais rápidos, acessíveis e consistentes. Este trabalho contribui para o futuro da integração entre LLMs e o desenvolvimento de software, destacando caminhos claros para uma utilização mais eficaz e eficiente dessas ferramentas.

8 REFERÊNCIAS

Referências

Gemini IA. "Guia de início rápido da API Gemini."Disponível em: https://ai.google.dev/gemini-api/docs/quickstart?hl=pt-brlang=python. Acesso em: 9 nov. 2024.

"Veja também o site oficial."Disponível em: https://gemini.google.com/app?hl=pt-BR. Acesso em: 9 nov. 2024.

Challa, N. (Srikrishna). "Essentials of Gemini — The new era of AI."Google Cloud - Community, 2024. Disponível em: https://medium.com/google-cloud/essentials-of-gemini-the-new-era-of-ai-efca53293341. Acesso em: 9 nov. 2024.

Masalkhi, M., Ong, J., Waisberg, E., & Lee, A. G. "Google DeepMind's Gemini AI versus ChatGPT: a comparative analysis in ophthalmology."*Eye*, vol. 38, pp. 1412–1417, 2024. Disponível em: https://www.nature.com/articles/s41433-024-02958-w. Acesso em: 9 nov. 2024.

Naseef, C. "Understanding the Impact of Large Language Models on Software Development."*Medium*, 2023. Disponível em: https://medium.com/@naseefcse/the-negative-impact-of-large-language-models-on-software-developers-a2e9dc1e2c69. Acesso em: 9 nov. 2024.

Wang, S., Ding, L., Shen, L., Luo, Y., Du, B., Tao, D. "OOP: Object-Oriented Programming Evaluation Benchmark for Large Language Models."Wuhan University, The University of Sydney, JD Explore Academy, 2023. Disponível em: https://arxiv.org/abs/2401.06628. Acesso em: 9 nov. 2024.