OnTask: um sistema para gerenciamento de projetos e tarefas

SOUZA NETO, Gualter Albino de [a] ; NOÉ, Paulo Ricardo Gregório[b]; TREVIZANO, Waldir Andrade[2]; BAIA, Joás Weslei[2]

gualteralbino1000@gmail.com
paulo.noe@unifagoc.edu.br

[a] Discente do curso de Bacharelado em Ciência da Computação do Centro Universitário Governador Ozanam
[b] Docente do Centro Universitário Governador Ozanam - UNIFAGOC

RESUMO

Pequenas empresas de desenvolvimento de software enfrentam desafios recorrentes na definição de prazos, estimativas de tempo e preservação do conhecimento sobre projetos, frequentemente devido à ausência de registros e documentação estruturada. Este trabalho propõe o desenvolvimento de um protótipo de API open source voltado para a gestão de projetos e atividades, com o objetivo de aprimorar o controle, a organização e a documentação dos processos internos dessas empresas. A metodologia incluiu a aplicação de uma pesquisa online exploratória com foco na coleta de dados quantitativos, contando com uma amostra de 396 desenvolvedores de software, para identificar os principais desafios de gestão enfrentados por pequenas empresas. Com base nos resultados, foi desenvolvido um protótipo seguindo uma arquitetura baseada nos princípios da Arquitetura Hexagonal, visando uma separação clara das camadas da aplicação para facilitar a manutenção, escalabilidade e personalização conforme as necessidades de cada empresa. O protótipo resultante oferece uma solução flexível e adaptável, facilitando o registro de atividades de um projeto e o uso de parâmetros personalizados de dificuldade e tempo para auxiliar no gerenciamento.

Palavras-chave: Controle de Atividades. Gestão de Projetos. Software de Gestão.

INTRODUÇÃO

O controle e gerenciamento eficazes são fundamentais para o sucesso de qualquer empreendimento. Como afirmado por Baldam, Valle e Rozenfeld (2014), o controle oferece informações cruciais aos decisores sobre o desempenho dos processos, sua conformidade com os planos estabelecidos, seu alinhamento com a estratégia organizacional e a necessidade de melhorias ou inovações. Nesse contexto, o gerenciamento de projetos assume um papel central, especialmente quando se busca alcançar metas de forma eficiente.

O gerenciamento de atividades em um projeto, segundo Barcellos, Rocha e Travassos (2004), é um elemento-chave para a eficácia operacional e o alcance de metas em qualquer organização. Esse gerenciamento envolve a coordenação de recursos, prazos e orçamentos para atingir resultados desejados dentro de um cronograma determinado. No setor de desenvolvimento de *software*, essa prática torna-se ainda mais crítica devido à natureza dinâmica e complexa dos projetos.

Sommerville(2019) destaca que o desenvolvimento de *software* requer uma sequência estruturada de etapas, como o entendimento do problema, levantamento de requisitos, modelagem, codificação, testes e, finalmente, manutenção até o fim do

ciclo de vida do *software*. Essas etapas precisam ser realizadas de maneira segura e objetiva para garantir a entrega de um *software* funcional. Ainda que utilizem metodologias bem definidas, as empresas frequentemente enfrentam dificuldades em estimar o tempo necessário para concluir atividades, especialmente pela ausência de registros e documentação dos processos anteriores.

A falta de documentação adequada pode impactar diretamente na qualidade e nos prazos dos projetos. De acordo com Barcellos, Rocha e Travassos (2004), entregar um produto de qualidade dentro dos prazos e custos previstos é um grande desafio para pequenas empresas. Este cenário evidencia a necessidade de melhorar a estimativa e o controle do tempo e custos dos projetos de *software*.

Além disso, Baldam, Valle e Rozenfeld (2014) ressaltam que pequenas empresas frequentemente enfrentam dificuldades para estabelecer processos internos claros e bem definidos. Esses processos acabam sendo gerenciados de forma inconsistente e não documentada, tornando-se parte do capital estrutural da organização de forma implícita. Essa situação é confirmada por estudos recentes, como o artigo *Balancing Quality and Delivery Speed: Methods for Agile Success* (2023), que identifica alguns motivos para a falta de documentação: a prioridade na entrega rápida e o custo e esforço de manutenção da documentação.

Outro problema correlato é a perda de conhecimento devido à rotatividade de membros da equipe. Frufrek (2015) afirma que a rotatividade de pessoal é um dos principais desafios para empresas de desenvolvimento de software. Santos (2017) complementa essa visão, afirmando que a sobrevivência das organizações depende da sua capacidade de compartilhar e reter conhecimento. Assim, a documentação se torna essencial para garantir a continuidade do software, mesmo com a saída de membros da equipe. Baldam, Valle e Rozenfeld (2014, p. 274) reforçam a importância dessa prática ao afirmar: "O que não é preenchido com informação correta, é completado pela imaginação".

Demonstrada a importância do tema, o problema abordado nesse trabalho é: Como as pequenas empresas de desenvolvimento de software realizam o gerenciamento de seus projetos e atividades?

Este trabalho como objetivo geral apresenta um protótipo de um API *open source* para auxiliar pequenas empresas de desenvolvimento de software no controle, mapeamento e documentação de seus projetos e atividades, visando melhorar a organização, a gestão do conhecimento e a eficiência operacional. Os objetivos específicos são: Desenvolver e aplicar um questionário online para coletar dados sobre os principais desafios enfrentados por pequenas empresas no processo de documentação de projetos e atividades; analisar os dados coletados para identificar e extrair insights, como os motivos mais comuns para a falta de documentação em projetos de software, a relação entre esse problema e o porte da empresa, e as formas mais utilizadas para estimativa de tempo nos projetos; Implementar um protótipo de uma API com as seguintes funcionalidades: Permitir o cadastro e a autenticação de usuários, garantindo acesso seguro ao sistema; Viabilizar o cadastro de grupos, que servirão como entidades centrais para o compartilhamento de parâmetros entre projetos associados a cada grupo; Permitir o cadastro de parâmetros para cada grupo, como níveis de dificuldade das tarefas, tempos estimados para as dificuldades e status de cada etapa, facilitando a gestão do

cronograma e dos processos; Habilitar o registro de projetos e a associação de atividades a esses projetos garantindo que cada atividade contenha detalhes como o que foi solicitado e o que realmente foi realizado; Permitir o convite de usuários para a participação em projetos específicos, promovendo a colaboração entre equipes; Implementar a funcionalidade de iniciar, pausar e finalizar atividades, com o registro de um histórico detalhado e o armazenamento do tempo gasto em cada atividade para controle preciso da execução.

REFERENCIAL TEÓRICO

Gestão de Projetos de Software

Sommerville (2011, p. 428) afirma que a gestão eficaz de projetos é crucial para o sucesso, garantindo que o projeto atenda e supere suas restrições, além de fornecer alta qualidade. Embora os critérios de sucesso possam variar de um projeto para outro, há elementos comuns, como a entrega dentro do prazo, o atendimento às expectativas dos clientes e a manutenção de uma equipe de desenvolvimento equilibrada e satisfeita. Esses fatores são especialmente importantes no contexto do desenvolvimento de software, onde os projetos costumam ser complexos e multifacetados.

O software, conforme mencionado por Sommerville (2011, p. 429), é um produto intangível, o que diferencia a gestão de projetos de *software* de outras áreas, como construção civil ou engenharia naval. Enquanto projetos físicos permitem que os gerentes visualizem o progresso e identifiquem atrasos de forma mais direta, os projetos de *software* exigem um acompanhamento mais abstrato. Os gerentes de projetos de software dependem de métricas, relatórios e indicadores de progresso para avaliar o andamento do trabalho, pois não é possível "ver" o software em construção da mesma maneira que se observa uma estrutura física. Essa intangibilidade do *software* torna a gestão de projetos ainda mais desafiadora, reforçando a necessidade de processos e ferramentas robustos para monitorar o progresso e mitigar riscos.

Além desses desafios, a ausência de documentação pode se tornar um obstáculo significativo na gestão de projetos de software. Segundo o estudo *Balancing Quality and Delivery Speed: Methods for Agile Success (2023)*, o foco na entrega rápida, característica central das metodologias ágeis, é uma das principais causas da falta de documentação formal em projetos de software. Embora a agilidade proporcione ciclos de desenvolvimento mais curtos e flexíveis, a redução nas práticas de documentação pode comprometer a rastreabilidade e a manutenção de longo prazo do software. Assim, encontrar um equilíbrio entre a velocidade de entrega e a qualidade, sem comprometer a documentação essencial, é um desafio que gestores de projetos precisam enfrentar.

Ferramentas de Gestão de Projetos e Atividades

Sommerville (2011, p. 452) destaca o papel crucial das ferramentas de gestão de projetos e atividades na otimização e organização das operações de uma empresa. Essas ferramentas oferecem uma variedade de benefícios, como a melhoria da eficiência organizacional, a gestão de tempo, e o apoio à tomada de decisões. Em um

ambiente de desenvolvimento de *software*, onde a complexidade e a dinâmica dos projetos são elevadas, tais ferramentas tornam-se ainda mais essenciais.

Barros e Calil (2019) conduziram um estudo de caso relevante sobre o uso do *software* Asana por pequenas empresas, evidenciando os benefícios da implementação dessas ferramentas pode trazer melhorias substanciais na gestão do tempo, nos processos e na adaptação a mudanças, resultando em decisões mais informadas e em um aumento significativo da produtividade. Isso, por sua vez, contribui para a melhoria dos resultados globais da empresa, demonstrando como as ferramentas de gestão de projetos podem ser um diferencial competitivo.

Keeling e Branco (2017) reforçam essa visão em seu livro "Gestão de Projetos". Eles argumentam que um gerenciamento eficaz de tarefas e projetos, facilitado por ferramentas de gestão, proporciona uma visão holística das atividades em andamento. Isso permite uma alocação mais eficiente de recursos, a identificação de gargalos e a realização de ajustes necessários para otimizar o desempenho. Além disso, essas ferramentas promovem uma colaboração mais eficaz entre os membros da equipe, facilitando a comunicação e garantindo que todos estejam alinhados com os objetivos do projeto. Essa comunicação clara e transparente é vital para a maximização da eficácia operacional e o sucesso dos projetos de *software*.

METODOLOGIA

Pesquisa e análise

Para o desenvolvimento deste trabalho, foi realizada uma pesquisa online utilizando o Google Forms como ferramenta de coleta de dados. O questionário, composto por sete perguntas fechadas, foi divulgado em comunidades de desenvolvedores de software no Facebook. A amostra foi classificada como não probabilística e de conveniência, dado que os participantes foram selecionados conforme sua disponibilidade e interesse em responder voluntariamente. Dessa forma, a pesquisa caracteriza-se como exploratória, com foco na coleta de dados quantitativos.

A coleta de dados ocorreu entre os dias 01 de julho de 2024 e 31 de agosto de 2024, resultando na participação de 396 indivíduos que se autodeclararam programadores atuantes em empresas de desenvolvimento de *software*. O objetivo principal da pesquisa foi obter dados numéricos sobre as práticas de gestão de projetos e atividades nessas empresas, buscando identificar, por meio de uma análise estatística, os principais problemas enfrentados por pequenas empresas de desenvolvimento de software.

Essa abordagem visa proporcionar um panorama geral das dificuldades comuns no gerenciamento de projetos e tarefas, servindo como base para o desenvolvimento do sistema proposto neste trabalho, o "OnTask". Ao definir os principais motivos que contribuem para a ocorrência de problemas, a pesquisa pretende contribuir para a melhoria das práticas de gestão no setor de *software*.

Modelagem do banco de dados

A modelagem do banco de dados é um elemento crucial para garantir uma compreensão abrangente do projeto, permitindo visualizar seu escopo e arquitetura. Para essa etapa, foi utilizada a ferramenta *MySQL Workbench*, amplamente

reconhecida por facilitar o design visual de bancos de dados e gerar diagramas claros e precisos das estruturas envolvidas, conforme descrito em sua documentação oficial (*MySQL Workbench Documentation*, 2024). As atividades dessa fase incluíram a modelagem do banco de dados, com a definição de entidades e relacionamentos, além da criação dos modelos conceitual e lógico para representar de forma precisa a estrutura de dados.

Para o armazenamento, optou-se pelo *PostgreSQL*, por questões de familiaridade, já que assim como os demais disponíveis ele é robusto e eficiente, sendo capaz de lidar com grandes volumes de dados e realizar consultas complexas de maneira eficiente.

Arquitetura do software

A arquitetura do sistema foi projetada seguindo os princípios da Arquitetura Hexagonal, proposta por Cockburn (2005), que visa a separação clara entre as camadas da aplicação. Essa abordagem facilita a manutenção e a evolução, promovendo um desenvolvimento mais sustentável. A Arquitetura Hexagonal, também conhecida como Arquitetura de Portas e Adaptadores, permite que o núcleo da aplicação seja isolado de implementações externas, como bancos de dados ou interfaces de usuário.

Robert C. Martin (2017), criador dos princípios SOLID, defende a adoção de boas práticas de design e desenvolvimento de software, como o próprio SOLID e DRY (D*on't Repeat Yourself*), que também são incorporadas neste projeto. Essas práticas, quando combinadas à arquitetura proposta, contribuem para a criação de um sistema robusto, flexível e altamente adaptável. Em um contexto de código *open source*, tais características são fundamentais, uma vez que facilitam a reutilização e a personalização da solução conforme as necessidades específicas das empresas usuárias.

Tecnologias para o desenvolvimento

Para o desenvolvimento do protótipo da API *open source* OnTask, foi utilizada a linguagem TypeScript, uma extensão do JavaScript que adiciona suporte a tipos estáticos e opcionais, trazendo maior segurança, legibilidade e escalabilidade ao desenvolvimento da aplicação. A escolha do TypeScript é estratégica, pois permite a utilização tanto no *back-end* quanto no *front-end* para aqueles que queiram implementar uma interface para interagir com a API, proporcionando uma experiência de desenvolvimento unificada e eficiente. Essa unificação facilita a integração da API com qualquer *front-end* personalizado ou existente, adequando-se às necessidades específicas das empresas que a adotarem. Conforme documentado na documentação oficial, *TypeScript Documentation* (2024), o uso de tipagem estática reduz a incidência de erros em tempo de execução e melhora a experiência do desenvolvedor, especialmente em projetos de maior escala.

Além disso, foi adotado o Node.js, uma plataforma que permite a execução de código JavaScript no lado do servidor. O Node.js é amplamente reconhecido por sua alta performance em aplicações web devido à sua arquitetura baseada em eventos e à abordagem não bloqueante de I/O, o que o torna ideal para a construção de aplicações escaláveis, particularmente em cenários que envolvem múltiplas requisições simultâneas ou operações I/O intensivas. Segundo a documentação

oficial, *Node.js Documentation* (2024), essa arquitetura contribui diretamente para a capacidade de gerenciar várias conexões concorrentes sem comprometer o desempenho, um fator crucial para APIs de alto tráfego como a proposta neste projeto.

Junto ao Node.js, foi utilizado o framework Express.js, uma das opções mais populares para a construção de aplicações web em Node.js. O Express.js se destaca pela simplicidade e flexibilidade, permitindo a criação de APIs robustas e modulares com uma estrutura mínima. De acordo com a documentação oficial, *Express Documentation (2024)*, o framework facilita a gestão de rotas, a integração de middleware e a utilização de bibliotecas complementares, garantindo uma arquitetura de *back-end* eficiente, escalável e de fácil manutenção. Esses fatores fazem do Express uma escolha sólida para a construção de APIs que exigem alto desempenho e seguem boas práticas de desenvolvimento, alinhadas aos requisitos de flexibilidade e extensibilidade do projeto OnTask.

RESULTADOS

Análise obtida com a pesquisa

A pesquisa teve como público alvo pequenas e médias empresas, durando cerca de um mês e contando com 396 respondentes que se declararam como desenvolvedores de software, e teve como validar os problemas mencionados e as possíveis causas, auxiliando na assertividade do problema e sua solução.

Conforme apresentado na Tabela 1, entre os 396 respondentes, apenas 131 (33,08%) afirmaram que sempre ou às vezes registram e documentam suas atividades em um projeto. Observa-se também uma relação direta entre o porte da empresa e a importância atribuída ao controle de atividades e processos. Dos 232 respondentes que trabalham em pequenas empresas (58,58% do total), apenas 2 (0,86%) indicaram que sempre ou às vezes realizam esse registro e documentação. Esses dados reforçam a observação de Baldam, Valle e Rozenfeld (2014), que destacam as dificuldades frequentes enfrentadas por pequenas empresas em estabelecer processos internos claros e bem definidos.

Tabela 1 – Relação entre o registro de informações e o porte da empresa

Porte da Empresa	Quantidade por Porte	Realizam o registro de informações: **Sempre/ As Vezes**	% Percentual
Pequeno	232	2	0,86%
Médio	122	87	71,31%
Grande	42	42	100,00%
Totalizador:	**396**	**131**	

Fonte: Análise com base na pesquisa realizada pelo autor (2024).

A Tabela 2 revela que o principal motivo para a falta de documentação ou registro de processos nas empresas, indicado por 308 respondentes (77,77% do total), é o uso de métodos arcaicos, como blocos de notas ou comunicação verbal. O segundo motivo mais citado, por 88 respondentes (22,22% do total), é a percepção de que a documentação é um processo complicado, custoso ou demorado com os

softwares disponíveis no mercado. Esses dados reforçam as conclusões de *Balancing Quality and Delivery Speed: Methods for Agile Success* (2023), que aponta o foco na entrega rápida como uma das principais razões para a ausência de documentação.

Tabela 2 – Justificativa das empresas não documentarem os processos

Porte da Empresa	Quantidade por Porte	Métodos arcaicos de gerenciamento da equipe/empresa	A empresa considera um processo complicado/custoso ao se utilizar de softwares como os presentes no mercado
Pequeno	232	188	67
Médio	122	120	21
Grande	42	0	0
Totalizador:	**396**	**308**	**88**
Percentual:		**77,77%**	**22,22%**

Fonte: Análise com base na pesquisa realizada pelo autor (2024).

Outro ponto que a pesquisa confirma é a perda de conhecimento com a saída de membros da equipe. Conforme indicado na Tabela 3, 355 respondentes (89,64% do total) afirmaram que essa situação é recorrente. Esses dados estão alinhados com a afirmação de Frufrek (2015), que relata a perda de conhecimento nas empresas devido à rotatividade de funcionários. É importante destacar que, entre os respondentes de pequenas e médias empresas, 354 (89,39% do total) declararam haver perda de conhecimento em 100% dos casos. Já em grandes empresas, esse problema foi identificado por apenas um respondente (2,38% do total).

Tabela 3 – Perda de conhecimento com a saída de membros da equipe.

Porte da Empresa	Quantidade por Porte	Perda de conhecimento da empresa com a saída de membros	% Percentual
Pequeno	232	232	100,00%
Médio	122	122	100,00%
Grande	42	1	2,38%
Totalizador:	**396**	**355**	
Percentual:		**89,64%**	

Fonte: Análise com base na pesquisa realizada pelo autor (2024).

A pesquisa também confirma a afirmação de Barcellos, Rocha e Travassos (2004), que destacam o grande desafio que pequenas empresas enfrentam para entregar um produto de qualidade dentro dos prazos e custos previstos. A dificuldade em cumprir os prazos pode ser observada na Tabela 4, onde 267 respondentes (67,42% do total) afirmaram que os prazos nunca são cumpridos. O segundo maior grupo, com 87 participantes (21,96% do total), declarou que os prazos são cumpridos ocasionalmente, enquanto apenas 42 respondentes (10,60% do total) indicaram que os prazos sempre são atendidos. A tabela também revela uma variação significativa de acordo com o porte da empresa: em grandes empresas, que contavam com 42 respondentes (10,60% do total), os prazos foram cumpridos sempre ou às vezes em 100% dos casos; em empresas de médio porte, representando 122 respondentes (30,80% do total), os prazos foram atendidos em 67,21% dos casos; por outro lado, em pequenas empresas, com 232 respondentes

(58,58% do total), os prazos foram cumpridos sempre ou às vezes em apenas 2,16% dos casos.

Tabela 4 – Cumprimento de prazos.

Porte da Empresa	Quantidade por Porte	Cumprem os prazos: Sempre	Cumprem os prazos: As vezes	Cumprem os prazos: Nunca	% Percentual
Pequeno	232	1	4	227	2,16%
Médio	122	0	82	40	67,21%
Grande	42	41	1	0	100,00%
Totalizador:	**396**	**42**	**87**	**267**	
Percentual		**10,60%**	**32,07%**	**57,32%**	

Fonte: Análise com base na pesquisa realizada pelo autor (2024).

Quanto à forma de definição dos prazos, a pesquisa revela que a abordagem mais utilizada, apontada por 231 respondentes (58,33% do total), é a realização de estimativas baseadas em palpites, sem embasamento em dados concretos. O segundo grupo, composto por 123 respondentes (31,09% do total), indicou que as estimativas são feitas pela gerência com base em sua experiência pessoal. Por fim, 42 respondentes (10,60% do total) afirmaram utilizar registros de tarefas anteriores para estimar os prazos, como mostrado na Tabela 5.

Tabela 5 – Formas utilizadas para estimativa de tempo

Porte da Empresa	Quantidade por Porte	Estimados através de palpites	Com base em deduções da gerência baseadas apenas	Com base em registros anteriores de tarefas
Pequeno	232	229	2	1
Médio	122	1	121	0
Grande	42	1	0	41
Totalizador:	**396**	**231**	**123**	**42**
Percentual:		**58,33%**	**31,09%**	**10,60%**

Fonte: Análise com base na pesquisa realizada pelo autor (2024).

A Tabela 5 também permite relacionar essas práticas com o porte das empresas. Nas pequenas empresas, 229 respondentes (98,71% do grupo) realizam estimativas por palpites, 2 respondentes (0,86%) baseiam-se na experiência da gerência, e 1 respondente (0,43%) utiliza registros de tarefas anteriores. Em empresas de médio porte, 121 dos 122 respondentes (99,18%) afirmaram que as estimativas são feitas com base na experiência da gerência, enquanto 1 respondente (0,82%) utiliza palpites sem embasamento em dados. Já nas grandes empresas, 41 dos 42 respondentes (97,62%) indicaram o uso de registros históricos para realizar as estimativas, e apenas 1 respondente (2,38%) baseia-se em palpites. **Protótipo de Software**

Modelagem do banco de dados

A Figura 1 em anexo representa o diagrama entidade-relacionamento do banco de dados, destacando a relação entre as 12 entidades, que incluem não

apenas usuários, atividades e projetos, mas também mecanismos de controle de permissões e histórico de atividades, garantindo uma gestão eficiente e flexível das tarefas no sistema.

Arquitetura

O desenvolvimento do projeto seguiu os princípios da Arquitetura Hexagonal (ou Arquitetura de Portas e Adaptadores), que organiza a aplicação em quatro camadas principais: *Adapters*, *Application*, *Domain* e *Infrastructure*. Essa divisão clara tem como objetivo isolar o núcleo da aplicação de suas dependências externas, como frameworks, bibliotecas e interfaces com outras aplicações.

Cada camada contém classes e interfaces padronizadas com a nomenclatura "Base", as quais servem como contratos que outros componentes devem implementar ou estender. Essa prática não só promove a consistência no design, como também centraliza trechos de código comuns, evitando duplicação e facilitando a manutenção ao longo do ciclo de vida do projeto.

A camada de domínio (*Domain*) é o núcleo do sistema, onde estão as regras de negócio e a lógica essencial. As outras camadas interagem com ela, mas não a afetam diretamente. A camada de aplicação (*Application*) coordena as operações e orquestra o fluxo de dados entre o domínio e os adaptadores externos. Já a camada de infraestrutura (*Infrastructure*) é responsável pelas implementações técnicas e pela integração com sistemas externos, como bancos de dados ou serviços de terceiros, enquanto a camada de adaptadores (*Adapters*) atua como intermediária entre as interfaces externas (APIs, interfaces de usuário) e o domínio.

Ao adotar essa abordagem, o projeto ganha maior flexibilidade e modularidade, permitindo que tecnologias e frameworks sejam substituídos sem afetar a lógica central da aplicação. Isso possibilita uma manutenção e evolução contínuas do sistema de maneira independente das ferramentas externas, garantindo uma maior longevidade do código.

O uso de classes base no projeto desempenha um papel fundamental ao garantir que operações e funcionalidades comuns não sejam repetidas, permitindo a reutilização de trechos de código de maneira eficiente e consistente em todas as camadas da aplicação. A separação por camadas é mantida, e cada camada é estruturada para herdar de classes base que centralizam comportamentos comuns, como gerenciamento de *ID*, *timestamps* e validações de dados. Isso não apenas promove a padronização e a organização do projeto, mas também facilita a manutenção, ao isolar responsabilidades e permitir que novos recursos sejam adicionados sem impacto direto em outras partes do sistema.

Um exemplo prático dessa abordagem pode ser visto no uso de decoradores (*decorators*) personalizados, criados especificamente para o projeto. Esses decoradores permitem definir, de forma simples e clara, regras de validação de propriedades obrigatórias nas classes de modelo. Como mostrado no Código 1, ao aplicar o "@BaseModel.Required" sobre uma propriedade, o sistema automaticamente a valida, garantindo que, ao instanciar ou salvar um modelo, os campos obrigatórios estejam preenchidos.

Código 1 – Exemplo de aplicação dos decoradores

```
export default class TipoStatusModel extends BaseModel<TipoStatusModel>
implements TTipoStatusModel {
@BaseModel.Required descricao:
string = '' @BaseModel.Required
private _tipo: string = ''
constructor(pObjeto: Partial<TipoStatusModel>, pValidarCadastro: boolean = true) {
super(pObjeto,pValidarCadastro) if
(pValidarCadastro) {
BaseModel.validate(this)
}
}
}
```

Fonte: produzido pelo autor (2024).

No exemplo de código da Código 1, a classe "TipoStatusModel" herda da classe base "BaseModel", o que garante que, ao ser instanciada, ela herdará automaticamente o comportamento de validação de propriedades obrigatórias. O decorador "@BaseModel.Required" facilita a declaração dessas regras diretamente nas propriedades, sem a necessidade de repetir a lógica de validação em diferentes pontos do sistema. Se um campo obrigatório, como "descricao" ou "tipo", não for preenchido, o método "BaseModel.validate" lançará um erro padronizado, informando o usuário sobre quais campos estão faltando, tornando o feedback claro e consistente.

Código 2 – Trecho de código exemplificando a classe "BaseModel"

```
export default abstract class BaseModel<T> { private _id!:
string
public criadoEm: Date = new Date()
public alteradoEm: Date = new Date() public
validarCadastro: boolean = true

static Required(target: any, propertyKey: string): void { if (!target) return
if (!target.constructor.requiredFields) target.constructor.requiredFields = []
target.constructor.requiredFields.push(propertyKey)
}
static Optional(target: any, propertyKey: string): void { if(!target.constructor.optionalFields)
target.constructor.optionalFields = [] target.constructor.optionalFields.push(propertyKey)
}
static validate(obj: any): void {
const requiredFields = obj.constructor.requiredFields || [] const missingFields =
requiredFields.filter((field: string) =>
!obj[field])
if (missingFields.length > 0) {
const fieldsTratados = missingFields.map((field: string) => { return field.replace('_', '')
```

```
})
throw new ValidationDomainError( obj.constructor.name,
'',
`Os seguintes campos são obrigatórios e estão vazios:
${fieldsTratados.join(', ')}`
)
}
}
}
```

Fonte: produzido pelo autor (2024).

Essa abordagem permite centralizar a lógica de validação e manter o código mais limpo, além de seguir princípios de *DRY* (*Don't Repeat Yourself*), uma prática fundamental em sistemas escaláveis e de fácil manutenção. Ao combinar a reutilização de classes base e a flexibilidade oferecida pelos decoradores, o projeto consegue lidar com validações de forma eficiente, garantindo que erros de preenchimento sejam interceptados de maneira clara e padronizada, independentemente da camada da aplicação em que ocorrem.

Na camada de aplicação (*Application*), as operações sobre o domínio são orquestradas. Os casos de uso (*use cases*) são definidos aqui, coordenando as interações entre as entidades do domínio e as outras camadas do sistema. Nesta camada, também são criadas classes de exceção para padronizar o tratamento de erros, garantindo consistência em toda a aplicação. Um elemento central nesta camada são os *DTOs* (*Data Transfer Objects*), que têm a função de padronizar a comunicação entre as camadas do sistema, como mostrado na Código 3. Os *DTOs* garantem que os dados sejam transferidos de maneira consistente e segura, além de simplificarem a validação e transformação de informações antes que sejam manipuladas pelo domínio.

Código 3 – Trecho de código exemplificando o DTO "CriarUsuarioDTO".

```
export default class CriarUsuarioDTO extends BaseDTO { @BaseDTO.Required
nome: string
@BaseDTO.Required role: string
@BaseDTO.Required email:
string @BaseDTO.Required
senha: string
constructor(pUsuario: UsuarioModel, pValidarCadastro: boolean = true) {
if (pValidarCadastro) BaseDTO.validate(this)
}
toDomain() {
return new UsuarioModel({nome: this.nome,role: this.role,email: this.email, senha: this.senha},
true).toObject()
}
```

}

Fonte: produzido pelo autor (2024).

Neste exemplo, a classe "CriarUsuarioDTO" herda de "BaseDTO" e utiliza decoradores

para definir campos obrigatórios, como "nome", "email", "role" e "senha". Ao instanciar um novo *DTO*, o construtor recebe um objeto do tipo "UsuarioModel" e realiza uma validação

automática dos campos. O método "toDomain()" permite a conversão do *DTO* para um objeto do domínio, integrando-o com o restante da aplicação.

Na camada adaptadora (*Adapters*), as interações externas são tratadas e a camada é dividida em duas partes principais: *HTTP* e *Postgres*. A parte *HTTP* lida com a interface de comunicação via *controllers*, implementada com *ExpressJS*. Aqui, os *handlers* atuam interceptando as comunicações entre os *controllers* e a camada de aplicação, validando e transformando os dados através dos *DTOs*, conforme mostrado no Código 4.

Código 4 – Trecho de código exemplificando a camada de adaptadores

```
export default class GrupoController { private handler:
GrupoHandler
private readonly logger = new Logger(this.constructor.name) constructor(pHandler:
GrupoHandler) {this.handler = pHandler} async incluir(req: Request,res: Response,next:
NextFunction):
Promise<void>{ try {
const body = req.body

const jwtPayload = req.sessao

const dto = new CriarGrupoDTO(body)

const sessao = new SessaoAuthDTO(jwtPayload)

const retorno = await this.handler.incluir(dto, sessao) res.status(201).send(retorno)
} catch (error) { this.logger.error(error) next(error)
}

}

}
```

Fonte: produzido pelo autor (2024).

A partição da camada de adaptadores (*Adapters*), *Postgres* é responsável por gerenciar

a persistência e recuperação de dados utilizando o *TypeORM*. Ela implementa os repositórios definidos na camada de domínio, garantindo que as operações de banco de dados estejam alinhadas aos contratos estabelecidos no domínio. Isso assegura que, independentemente da tecnologia de banco de dados utilizada, as regras e retornos sejam sempre consistentes e centralizados, conforme demonstrado no Código 5.

Código 5 – Trecho de código exemplificando a camada de adaptadores.

```
export default class UsuarioPostgresRepository implements UsuarioRepository {
private readonly logger: Logger

private readonly usuarioRepository: Repository<UsuarioEntity> constructor() {
this.logger = new Logger(this.constructor.name) const postgresConfig =
PostgresConfig.getInstance() this.usuarioRepository =
postgresConfig.getDataSource().getRepository(UsuarioEntity)

}

async incluir(pRegistro: UsuarioModel): Promise<UsuarioModel> { try {
const usuario = await this.usuarioRepository.save(pRegistro) return new
UsuarioModel(usuario)
} catch (error) { this.logger.error(error) throw
error
}

}

}
```

Fonte: produzido pelo autor (2024).

Nesta implementação, o repositório "UsuarioPostgresRepository" se conecta ao banco

de dados por meio de uma instância *Singleton* de "PostgresConfig", que facilita o gerenciamento da conexão com o banco de dados. A operação de inclusão (incluir) converte o modelo de domínio (UsuarioModel) em uma entidade persistível e utiliza o *TypeORM* para salvar a instância no banco de dados. Se ocorrer um erro durante o processo, ele é capturado e logado, garantindo que o fluxo de exceções seja tratado de forma adequada.

A camada de infraestrutura é responsável por configurar e gerenciar as tecnologias e ferramentas que suportam a execução da aplicação, incluindo bancos de dados, bibliotecas e serviços externos. Um dos princípios chave desta camada é que ela não interage diretamente com as regras de negócio, que estão isoladas nas camadas de **domínio** e **aplicação**. Isso garante a separação de responsabilidades, promovendo uma arquitetura limpa, modular e de fácil manutenção, como mostrado no Código 6.

Código 6 – Trecho de código exemplificando a inicialização da aplicação e das configurações de cada tecnologia

```
async function IniciarServer() { try {
logger.info('Iniciando o servidor...')

const postgres = PostgresConfig.getInstance() await postgres.start()
const express = new ExpressConfig() await express.start()
const swagger = new SwaggerConfig(express.app) await swagger.start()
logger.info('Servidor iniciado com sucesso!')
```

```
} catch (error) {
logger.error(`Falha ao iniciar o servidor: ${error}`)
}
}
IniciarServer()
```

Fonte: produzido pelo autor (2024).

Cada uma das instâncias (*PostgresConfig*, *ExpressConfig*, *SwaggerConfig*) contém suas respectivas configurações. O *PostgresConfig* gerencia a conexão com o banco de dados PostgreSQL, enquanto o *ExpressConfig* configura o servidor web. Por fim, o *SwaggerConfig* integra a documentação interativa da API, facilitando o acesso e o teste dos *endpoints*.

Essa abordagem modular na inicialização do servidor demonstra a importância da camada de infraestrutura em garantir que todos os serviços estejam corretamente configurados e funcionando, sem que detalhes de implementação técnica vazem para outras partes do sistema.

CONCLUSÃO

A pesquisa realizada com 396 desenvolvedores de software, incluindo profissionais de pequenas, médias e grandes empresas, confirmou que pequenas empresas enfrentam maiores dificuldades na gestão de seus projetos. A pesquisa identificou uma relação entre a incidência de problemas de gestão e a falta de registro e documentação, sendo possível observar que, quanto menor o porte da empresa, maior é a frequência desses problemas. Dos 232 respondentes que trabalham em pequenas empresas (58,58% do total), apenas 2 (0,86%) indicaram que sempre registram e documentam atividades em projetos. Em contraste, nas médias e grandes empresas, o índice de registro de atividades é superior a 70%, conforme apresentado na Tabela 1. Os principais motivos para a falta de registro foram apontados por 308 respondentes (77,77%): o uso de métodos arcaicos ou informais, como blocos de notas ou comunicação verbal. Outros 88 respondentes (22,22%) justificaram a ausência de documentação pela percepção de que este é um processo complexo, custoso ou demorado, conforme a Tabela 2.

Esses fatores impactam diretamente a capacidade das pequenas empresas de planejar e executar projetos de maneira organizada. Ao analisar o cumprimento de prazos, observou- se que, dos 232 respondentes de pequenas empresas, apenas 5 afirmaram que os prazos são sempre ou ocasionalmente cumpridos, representando um índice de cumprimento de apenas 2,16%, conforme visto na Tabela 4.

Com base nesses resultados foi criado um protótipo de uma API *open source*, chamada **OnTask**, que visa fornecer às pequenas empresas uma ferramenta flexível para o gerenciamento de suas atividades, permitindo o cadastro de projetos, atividades, parâmetros de dificuldade com associação de tempo e aproveitamento de parâmetros por grupos. A arquitetura adotada, baseada nos princípios da Arquitetura Hexagonal, permite que o núcleo do sistema esteja isolado das

dependências externas, facilitando a adaptação e evolução da tecnologia ao longo do tempo, e abrindo a possibilidade para que empresas personalizem a solução de acordo com suas necessidades. Este protótipo desenvolvido está disponível como um repositório *open source* no GitHub no link: https://github.com/GualterAlbino/OnTaskAPI, contendo toda a documentação detalhada das rotas da API e instruções para instalação e uso. Isso garante que empresas e desenvolvedores interessados possam facilmente acessar, utilizar e até contribuir para a evolução do projeto, ampliando sua utilidade.

Com o intuito de evoluir a ferramenta, pode-se destacar algumas sugestões de trabalhos futuros: (1) desenvolvimento de uma interface gráfica própria para uso direto, além da API; (2) implementação de relatórios padrão para exibir informações resumidas sobre tempo e atividades dos projetos, com opções de personalização; e (3) criação de testes automatizados para assegurar maior confiabilidade e robustez.

REFERÊNCIAS

Balancing Quality and Delivery Speed: Methods for Agile Success. 2023. Disponível em: https://www.leanwisdom.com/blog/balancing-quality-and-delivery-speed-methods-for-agile- success. Acesso em 31 ago. 2024.

BALDAM, R., VALLE R., & ROZENFELD, H. (2014). **Gerenciamento de processos de negócio BPM: uma referência para implantação prática** (1a ed.). Rio de Janeiro: Elsevier.

BARCELLOS, M. P.; ROCHA, A. R.; TRAVASSOS, G. H (2004). **Planejamento de Custos em Ambientes de Desenvolvimento de Software Orientados à Organização**. In: Anais do III Simpósio Brasileiro de Qualidade de Software. SBC, 2004. p. 366-380.

BARROS R. S. A. D.; CALIL, R. (2019). **Um estudo de caso de pequenas empresas que utilizam o gerenciador de tarefas Asana.**

COCKBURN, Alistair. **Hexagonal Architecture** (2005). Disponível em: https://alistair.cockburn.us/hexagonal-architecture/ . Acesso em: 30 set. 2024.

Express Documentation. 2024. Disponível em: https://expressjs.com/en/5x/api.html. Acesso em 9 out. 2024.

FRUFREK, G. L. **Um estudo sobre a rotatividade de pessoal entre profissionais de empresas brasileiras de desenvolvimento de software**. 2015. Dissertação de Mestrado. Universidade Tecnológica Federal do Paraná.

KEELING, R.; BRANCO, R. H. F. **Gestão de projetos**. Saraiva Educação SA, 2017.

MARTIN, Robert C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. 1. ed. Boston: Prentice Hall, 2017.

SANTOS, M. J. F. N. dos; WANE, R. **Estratégia para Evitar a Fuga de Conhecimento Organizacional: o caso da ALSTOM Portugal.** Desenvolvimento Socioeconômico em Debate.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.
SOMMERVILLE, I. **Engenharia de software**. 10. ed. São Paulo: Editora Pearson, 2019.

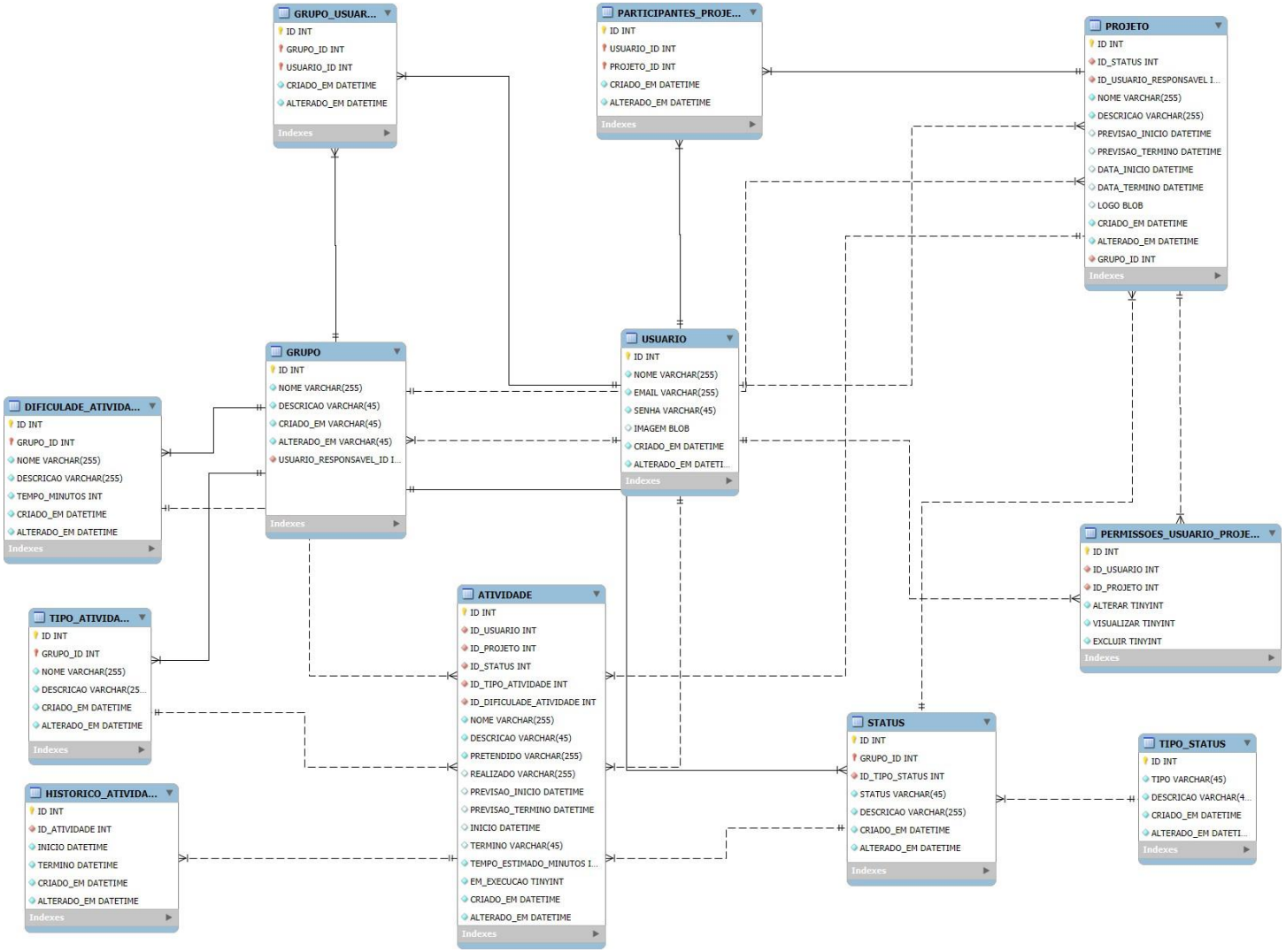
My SQL Workbench Documentation. 2024. Disponível em: https://www.mysql.com/products/workbench/. Acesso em 22 abr. 2024.

Node.js Documentation. 2024. Disponível em: https://nodejs.org/docs/latest/api/. Acesso em 22 abr. 2024.

PostgreSQL Documentation. 2024. Disponível em: https://www.postgresql.org/docs/. Acesso em 22 abr. 2024.

Typescript Documentation. 2024. Disponível em: https://www.typescriptlang.org. Acesso em 22 abr. 2024.

Figura 1 – Modelo entidade relacionamento



Fonte: dados da pesquisa (2024).