UNIVERSIDADE DE ARARAQUARA

PLAN-IT GERENCIADOR DE TAREFAS WEB

PLAN-IT WEB TASK MANAGEMENT SYSTEM

PLAN-IT GESTOR DE TAREAS WEB

Caio Henrique Feiria[1], Felipe Diniz Dallilo[2], Fabiana Florian[3]

https://doi.org/10.47820/recima21.v6i1.7025

PUBLICADO: 11/2025

RESUMO

A desorganização de tarefas em ferramentas inadequadas, como e-mails e planilhas, resulta frequentemente em perda de prazos, estresse e queda na produtividade. Este trabalho objetivou desenvolver o "Plan-It", um sistema web para gerenciamento de tarefas e projetos, focado em otimizar a organização de pequenas equipes e usuários individuais. A metodologia utilizada foi a aplicada, com o desenvolvimento de uma arquitetura em três camadas, utilizando Angular para o *frontend* , Spring Boot para o *backend* e MySQL para a persistência de dados. A qualidade do *software* foi aferida com testes unitários (JUnit), análise de cobertura de código (JaCoCo) e documentação de API (Swagger). O resultado é uma plataforma funcional que centraliza informações, permite a clara definição de responsáveis e prazos e otimiza o fluxo de trabalho. Conclui-se que o sistema valida a hipótese de que uma ferramenta organizada melhora a gestão de atividades, oferecendo controle e clareza, superando as limitações das ferramentas genéricas.

PALAVRAS-CHAVE: Angular. Gestão de Tarefas. Produtividade. Spring Boot. Sistema Web.

ABSTRACT

Task disorganization in inadequate tools, such as emails and spreadsheets, often results in missed deadlines, stress, and decreased productivity. This paper aimed to develop "Plan-It," a web-based task and project management system focused on optimizing organization for small teams and individual users. The methodology used was applied, developing a three-tier architecture using Angular for the frontend, Spring Boot for the backend, and MySQL for data persistence. Software quality was measured with unit tests (JUnit), code coverage analysis (JaCoCo), and API documentation (Swagger). The result is a functional platform that centralizes information, allows clear definition of responsibilities and deadlines, and optimizes workflow. It is concluded that the system validates the hypothesis that an organized tool improves activity management by offering control and clarity, overcoming the limitations of generic tools.

KEYWORDS: *Angular. Productivity. Spring Boot. Task Management. Web System.*

RESUMEN

La desorganización de tareas en herramientas inadecuadas, como correos electrónicos y hojas de cálculo, resulta frecuentemente en pérdida de plazos, estrés y disminución de la productividad. Este trabajo tuvo como objetivo desarrollar "Plan-It", un sistema web para la gestión de tareas y proyectos, enfocado en optimizar la organización de pequeños equipos y usuarios individuales. La metodología utilizada fue aplicada, con el desarrollo de una arquitectura de tres capas, empleando Angular para el frontend, Spring Boot para el backend y MySQL para la persistencia de datos. La calidad del software fue evaluada mediante pruebas unitarias (JUnit), análisis de cobertura de código (JaCoCo)

[1] Graduando do Curso de Sistemas de Informação Caio Henrique Feiria da Universidade de Araraquara - UNIARA. Araraquara-SP.
[2] Orientador: Docente do curso de Sistemas de Informação Felipe Diniz Dallilo da Universidade de Araraquara - UNIARA. Araraquara-SP.
[3] Coorientador: Docente do curso de Sistemas de Informação Fabiana Florian da Universidade de Araraquara - UNIARA. Araraquara-SP.

y documentación de API (Swagger). El resultado es una plataforma funcional que centraliza información, permite la definición clara de responsables y plazos, y optimiza el flujo de trabajo. Se concluye que el sistema valida la hipótesis de que una herramienta organizada mejora la gestión de actividades, proporcionando control y claridad, superando las limitaciones de las herramientas genéricas.

PALABRAS CLAVE: *Angular. Gestión de Tareas. Productividad. Spring Boot. Sistema Web.*

1. INTRODUÇÃO

Organizar tarefas é um desafio comum em empresas, universidades e na vida pessoal. A ausência de uma abordagem estruturada para o gerenciamento de atividades resulta em prazos perdidos, indefinição de responsabilidades e queda na qualidade do trabalho. David Allen (2015, p. 27) ilustra este ponto ao citar a frase de David Kekich, afirmando que “a ansiedade é consequência da falta de controle, de organização, de preparação e de ação”, o que evidencia como a desorganização compromete a produtividade e aumenta os níveis de estresse. Em equipes pequenas, a desorganização causa retrabalho e desânimo, e para estudantes, prejudica os resultados nos estudos e gera mais pressão.

Frequentemente, recorre-se a ferramentas inadequadas para essa finalidade, como e-mails, aplicativos de mensagens ou planilhas. Tais métodos são pouco práticos e difíceis de ajustar às necessidades de equipes ou usuários individuais. A dispersão das informações em vários lugares dificulta a priorização e o acompanhamento das demandas, o que gera atrasos, confusões e problemas no trabalho colaborativo.

Diante desse cenário, o presente trabalho propõe o desenvolvimento de um sistema web para a gestão de tarefas e projetos. A solução visa centralizar as informações, definir responsáveis e prazos de forma clara e destacar as prioridades, com o objetivo de otimizar a produtividade e a satisfação de pequenas equipes e usuários individuais.

A dificuldade em organizar tarefas utilizando ferramentas atuais, como e-mails e planilhas, justifica este projeto, pois as informações espalhadas atrapalham o trabalho e aumentam o estresse. Muitos usuários, incluindo estudantes e pequenos times, enfrentam dificuldades para identificar responsabilidades e cumprir prazos, gerando confusão e resultados insatisfatórios.

A justificativa do projeto é reforçada por Allen (2015, p. 33), que aponta a necessidade de “capturar todas as coisas que devem ser feitas [...] num sistema lógico e confiável, fora da sua cabeça”. Um trabalho organizado que ajude a manter as tarefas claras e fáceis de gerenciar é, portanto, essencial. Este projeto é importante por propor uma solução prática para um problema comum, beneficiando equipes e pessoas que buscam mais produtividade e menos estresse.

A dificuldade observada é que a ausência de uma plataforma com as características necessárias de organização e centralização faz com que muitos usuários enfrentem a dispersão de informações, o que atrapalha o trabalho e aumenta o estresse. A falta de clareza sobre quem é responsável por cada tarefa e a dificuldade em cumprir prazos reduzem a eficiência e causam desânimo e ansiedade.

Propõe-se, assim, uma aplicação intuitiva e organizada, cuja hipótese é que sua eficácia na melhoria da gestão de atividades será comprovada pela capacidade de oferecer controle e clareza

sobre os compromissos. A validação dessa hipótese se dará por meio de testes de usabilidade e coleta de *feedback.*

A metodologia adotada para o desenvolvimento do sistema é de natureza aplicada, envolvendo a construção da solução com o uso de tecnologias atuais. As tecnologias escolhidas para a implantação incluem angular para o *frontend*, Spring Boot para o *backend* e MySQL para a persistência dos dados. O estudo será conduzido em um contexto acadêmico, com etapas de validação que envolverão o público-alvo.

2. REVISÃO BIBLIOGRÁFICA

Esta seção apresenta os fundamentos do desenvolvimento de sistemas web, com ênfase nas tecnologias adotadas neste projeto. Além disso, aborda metodologias de gestão de tarefas e produtividade, alinhadas ao contexto do sistema proposto.

2.1. Gestão de Tarefas e Produtividade

A gestão de tarefas é um processo sistemático que abrange o planejamento, a organização e o monitoramento de atividades, tanto em contextos individuais quanto coletivos. Essa abordagem visa garantir o cumprimento de prazos e a manutenção dos padrões de qualidade, o que é alcançado por meio da definição clara de responsabilidades e do acompanhamento contínuo do progresso.

A produtividade, por sua vez, está intrinsecamente ligada à eficácia desse gerenciamento. Segundo Allen (2015), a mente se torna mais clara e criativa quando liberada da sobrecarga de gerenciar pendências, um princípio central do método

Getting Things Done (GTD). Esse sistema, estruturado em cinco etapas — capturar, esclarecer, organizar, refletir e engajar — converte demandas complexas em ações gerenciáveis, o que resulta na redução da ansiedade e na potencialização do foco.

Dessa forma, a organização sistemática defendida pelo autor, quando aliada a técnicas de priorização, não apenas previne atrasos, mas também otimiza o uso do tempo e da energia, elementos cruciais para a produtividade em todas as esferas da vida.

2.2. Programação Orientada a Objetos

Este trabalho adota como base técnica a Programação Orientada a Objetos (POO), seguindo os princípios estabelecidos por Gamma, Helm, Johnson e Vlissides. Conforme os autores, “programas orientados a objetos são feitos de objetos”, onde cada objeto encapsula tanto os dados quanto os procedimentos que operam sobre eles. Essa abordagem permite representar entidades do mundo real como unidades modulares interativas, organizadas através de classes (modelos abstratos), métodos (operações) e atributos (estados internos) (GAMMA *et al*., 1994, p. 27).

Classes: representam os elementos fundamentais da programação orientada a objetos, atuando como modelos que estabelecem tanto a interface pública quanto a estrutura interna dos objetos, determinando não apenas a representação dos dados encapsulados, mas também definindo o conjunto completo de operações que suas instâncias podem realizar (GAMMA *et al*., 1994).

Métodos: representam as ações associadas a uma classe. Eles definem operações que os objetos podem realizar, como *save* () para persistir dados no banco ou *update* () para atualizar os dados persistidos em banco. Diferentemente de funções convencionais, os métodos estão intrinsecamente vinculados aos objetos, operando sobre seus estados internos.

Atributos: os atributos representam as propriedades fundamentais que caracterizam um objeto na programação orientada a objetos, servindo como variáveis internas que armazenam os dados essenciais de uma instância de classe. Por exemplo, na classe Projeto, os atributos nome (do tipo *String*), prazo (tipo *LocalDate*) e prioridade (tipo *Enum*) definem suas características principais, sendo que o conjunto específico de valores atribuídos a essas propriedades em determinado momento constitui o estado atual do objeto (GAMMA *et al.,* 1994).

Na Programação Orientada a Objetos, os objetos se comunicam trocando mensagens através da chamada de métodos. Em um sistema de gerenciamento de tarefas, por exemplo, quando um objeto Usuário chama o método criar Tarefa (), ele interage com outros objetos como Tarefa e Repositório para processar a solicitação. Essa abordagem promove o baixo acoplamento entre componentes, facilitando a manutenção e evolução do sistema, conforme os princípios estabelecidos por Gamma *et al.* (1994). A modularização resultante melhora significativamente a organização e qualidade do código.

2.3. Arquitetura do Sistema: Três Camadas

A arquitetura em três camadas é um modelo consolidado na Engenharia de *Software* para a construção de aplicações, visando separar as responsabilidades do sistema. Essa abordagem estrutural é justificada por especialistas como SOMMERVILLE (2019) por proporcionar escalabilidade, desempenho e maior facilidade de manutenção.

A arquitetura define três níveis distintos, garantindo que a alteração em uma camada não impacte as demais:

• Apresentação *(Frontend):* Responsável pela interface com o usuário, implementada neste projeto utilizando Angular.

• Lógica do Aplicativo *(Backend):* Onde reside a lógica de negócio, implementada com Spring Boot, que hospeda a API RESTful.

• Camada de Dados: Responsável pela persistência e recuperação de informações, utilizando o MySQL.

A distinção clara das responsabilidades entre a lógica do aplicativo e a camada de dados é essencial para a alta coesão do sistema.

2.4. Angular

Angular é um *framework* de código aberto desenvolvido pelo Google em 2010 para a criação de aplicativos dinâmicos e interativos da web. Sendo um *framework* escolhido por muitas empresas e pessoas desenvolvedoras devido à sua robustez e conjunto de recursos abrangente. Sua estrutura modularizada promove a reutilização de código e a manutenção mais fácil.

De acordo com Batista (2025), ele traz poderosas ferramentas para lidar com tarefas complexas, como:

- Componentização;
- Ciclos de vida;
- *Data binding*;
- Manipulação de formulários;
- Roteamento.

2.5. TypeScript

> O TypeScript é um pré-processador *(superset)* de códigos JavaScript open source desenvolvido e mantido pela Microsoft. Ele foi desenvolvido pelo time do Anders Hejlsberg, arquiteto líder do time TypeScript e desenvolvedor do C#, Delphi e do Turbo Pascal. A sua primeira versão, a 0.8, foi lançada em 1 de outubro de 2012. (ADRIANO, 2021, p. 7).

A principal vantagem do TypeScript está na sua tipagem estática, que ajuda a prevenir erros durante o desenvolvimento, identificando potenciais bugs antes mesmo da execução do código. Essa abordagem traz mais segurança e robustez aos projetos, especialmente em aplicações complexas, permitindo que os desenvolvedores detectem e corrijam problemas ainda na fase de codificação, algo que não seria possível com JavaScript puro. A tipagem também melhora a manutenibilidade e escalabilidade do código, facilitando a colaboração em equipe e o desenvolvimento de sistemas mais estruturados.

Como os navegadores não interpretam código TypeScript, conforme explica Adriano (2021), "precisamos transpilar o nosso código para uma das versões ECMAScript". Esse processo de compilação é realizado pelo compilador do TypeScript, que converte o arquivo fonte .ts em código JavaScript executável .js.

2.6. Spring Boot (Java 21)

O Spring Boot configura-se como principal *framework* Java para desenvolvimento de APIs RESTful, integrando produtividade e robustez técnica (ACELIN, 2023; SPRING, 2025). Desenvolvido como parte do ecossistema Spring, ele facilita a criação de aplicações autônomas *(stand-alone)* e prontas para produção. Sua arquitetura opinativa e o uso de dependências *'starter'* simplificam a configuração de *build* e permitem iniciar projetos com o mínimo de esforço, não exigindo geração de código nem configuração XML.

Dentre seus principais benefícios para a construção de APIs REST, destacam-se:

• Arquitetura MVC simplificada: Utiliza anotações intuitivas, como @RestController, @GetMapping e @PostMapping, para o mapeamento rápido de *endpoints.*

• Integração com Spring Data JPA: Permite a implementação automática de operações CRUD básico, consultas personalizadas com @Query, além de paginação e ordenação nativas.

• Validação declarativa robusta: Suporta @Valid para validação de objetos e anotações específicas (@Size, @Pattern) para regras de negócio, com mensagens de erros personalizáveis.

Conclui-se que o *framework* é particularmente eficaz para APIs corporativas que demandam segurança, transacionalidade e integração com sistemas heterogêneos, sem comprometer a velocidade de desenvolvimento.

2.7. Qualidade e JUnit

A qualidade de um *software* depende do domínio de boas práticas de programação, que asseguram legibilidade e manutenção, e do uso de ferramentas de avaliação contínua, como o JUnit, para automação de testes (AZEVEDO, 2018). Os testes são fundamentais para detectar falhas precocemente, reduzir custos com correções tardias e aumentar a confiança no produto final.

O JUnit é uma ferramenta *open-source* em Java que ajuda a criar e executar testes automáticos para classes e métodos. Além disso, ele permite rodar todos os testes de uma vez sempre que o código muda, garantindo que o sistema funcione corretamente e sem falhas, promovendo maior confiança no desenvolvimento e na entrega final, como enfatizado por Azevedo (2018).

Vantagens de utilizar o JUnit

- Automação de testes: executa toda a bateria de testes com um único comando.
- *Feedback* imediato: identifica falhas logo após alterações no código.
- Segurança em mudanças: garante que as refatorações não quebrem funcionalidades existentes.

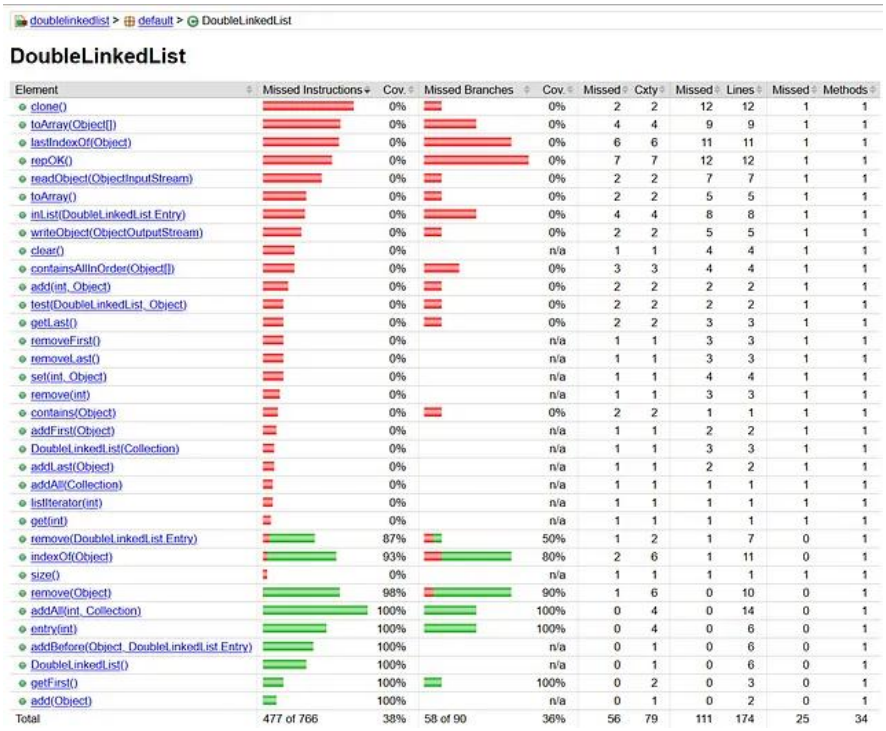
2.7.1. JaCoCo Coverage

O JaCoCo (Java Code Coverage) é uma biblioteca *open-source* que mede o quanto do código-fonte Java é executado pelos testes automatizados. Ele gera relatórios detalhados que mostram a porcentagem de cobertura geral, bem como indica exatamente quais linhas, métodos e classes foram testados ou não. Com esses dados, é possível identificar trechos de código sem testes, ajudando equipes de desenvolvimento a melhorar a robustez do *software* e garantir que novas alterações não deixem funcionalidades importantes sem verificação.

A cobertura de código corresponde à porcentagem de linhas executadas por testes automatizados, indicando diretamente quais trechos do programa permanecem sem verificação. Assim, um índice de 90 % revela que 10 % do código não foi alcançado pelos testes, apontando áreas que podem requerer novos casos de verificação. É importante observar que atingir 100 % de cobertura significa apenas que todas as linhas foram executadas em algum teste, sem implicar necessariamente na validação de todos os cenários possíveis (VERÍSSIMO, 2020).

O relatório do JaCoCo destaca, de forma visual e navegável, exibindo os trechos de código cobertos e não cobertos com a sua porcentagem dos testes, permitindo a identificação imediata de lacunas na suíte de testes (Figura 1). Essa precisão agiliza a correção de falhas e o reforço de cenários críticos, contribuindo para o aumento contínuo da qualidade do *software*.

Figura 1. Relatório de cobertura do código gerado pelo JaCoCo



doublelinkedlist > default > DoubleLinkedList

DoubleLinkedList

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
clone()		0%		0%	2	2	12	12	1	1
toArray(Object[])		0%		0%	4	4	9	9	1	1
lastIndexOf(Object)		0%		0%	6	6	11	11	1	1
repOK()		0%		0%	7	7	12	12	1	1
readObject(ObjectInputStream)		0%		0%	2	2	7	7	1	1
toArray()		0%		0%	2	2	5	5	1	1
inList(DoubleLinkedList.Entry)		0%		0%	4	4	8	8	1	1
writeObject(ObjectOutputStream)		0%		0%	2	2	5	5	1	1
clear()		0%		n/a	1	1	4	4	1	1
containsAllInOrder(Object[])		0%		0%	3	3	4	4	1	1
add(int, Object)		0%		0%	2	2	2	2	1	1
test(DoubleLinkedList, Object)		0%		0%	2	2	2	2	1	1
getLast()		0%		0%	2	2	3	3	1	1
removeFirst()		0%		n/a	1	1	3	3	1	1
removeLast()		0%		n/a	1	1	3	3	1	1
set(int, Object)		0%		n/a	1	1	4	4	1	1
remove(int)		0%		n/a	1	1	3	3	1	1
contains(Object)		0%		0%	2	2	1	1	1	1
addFirst(Object)		0%		n/a	1	1	2	2	1	1
DoubleLinkedList(Collection)		0%		n/a	1	1	3	3	1	1
addLast(Object)		0%		n/a	1	1	2	2	1	1
addAll(Collection)		0%		n/a	1	1	1	1	1	1
listIterator(int)		0%		n/a	1	1	1	1	1	1
get(int)		0%		n/a	1	1	1	1	1	1
remove(DoubleLinkedList.Entry)		87%		50%	1	2	1	7	0	1
indexOf(Object)		93%		80%	2	6	1	11	0	1
size()		0%		n/a	1	1	1	1	1	1
remove(Object)		98%		90%	1	6	0	10	0	1
addAll(int, Collection)		100%		100%	0	4	0	14	0	1
entry(int)		100%		100%	0	4	0	6	0	1
addBefore(Object, DoubleLinkedList.Entry)		100%		n/a	0	1	0	6	0	1
DoubleLinkedList()		100%		n/a	0	1	0	6	0	1
getFirst()		100%		100%	0	2	0	3	0	1
add(Object)		100%		n/a	0	1	0	2	0	1
Total	477 of 766	38%	58 of 90	36%	56	79	111	174	25	34

Fonte: Veríssimo (2020)

2.7.2. Swagger e a Documentação de APIs

Para Martins (2022) a documentação de APIs é crucial para a qualidade de *software*, pois simplifica a integração entre sistemas e a comunicação entre desenvolvedores e testadores. O Swagger destaca-se como uma ferramenta amplamente utilizada para criar especificações claras de interfaces de programação de aplicações (APIs), descrevendo endpoints, parâmetros, métodos HTTP e respostas em formatos como YAML ou JSON, o que promove padronização e legibilidade.

O Swagger facilita muito o trabalho com APIs usando anotações e o Swagger UI. As anotações, colocadas direto no código em linguagens como Java, descrevem como a API funciona, tornando a documentação mais prática. O Swagger UI cria uma página visual no navegador onde dá para testar e entender os endpoints da API sem complicação.

2.8. MySQL

O MySQL consolida-se como um dos sistemas de gerenciamento de banco de dados (SGBD) relacionais mais utilizados globalmente, destacando-se por sua natureza de código aberto, desempenho otimizado e confiabilidade (ERICKSON, 2023). Desenvolvido originalmente pela MySQL AB e atualmente mantido pela Oracle Corporation, esse SGBD desempenha um papel fundamental no armazenamento e recuperação de dados estruturados, sendo especialmente eficiente para operações CRUD *(Create, Read, Update, Delete)* em aplicações web.

Entre seus principais benefícios, destacam-se: performance otimizada para consultas frequentes, suporte a transações ACID (garantindo integridade dos dados), escalabilidade horizontal (via replicação e sharding) e segurança robusta com criptografia e controle de acesso granular (ERICKSON, 2023). Além disso, o MySQL oferece suporte a dados semiestruturados (JSON), atendendo às demandas modernas de aplicações flexíveis.

3. DESENVOLVIMENTO

Esta seção apresenta o processo de construção e implementação do sistema de gerenciamento de tarefas "Plan-It". Partindo dos fundamentos teóricos e das tecnologias abordadas na Revisão Bibliográfica, o foco aqui é demonstrar a aplicação prática dos conceitos para transformar os requisitos do projeto em uma solução funcional.

A implementação segue uma arquitetura em três camadas, que organiza o sistema em níveis distintos e com responsabilidades bem definidas.

Para isso, foram utilizadas as seguintes tecnologias principais: Angular para o desenvolvimento da interface com o usuário *(frontend),* Spring Boot para a implementação da lógica de negócio e da API RESTful *(backend),* e MySQL como sistema de gerenciamento para a persistência dos dados.

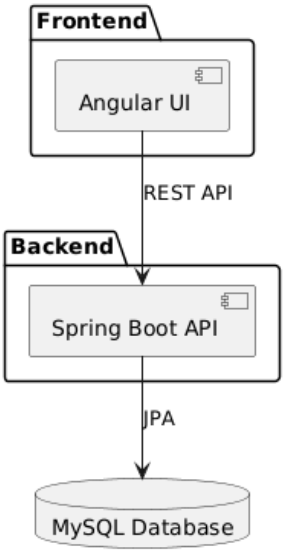
3.1 Arquitetura do Sistema

A arquitetura do sistema foi projetada segundo o padrão em camadas, separando claramente as responsabilidades de apresentação, negócio e persistência de dados. Essa abordagem estrutural é justificada por especialistas como SOMMERVILLE (2019) por proporcionar escalabilidade, desempenho e maior facilidade de manutenção.

Para o desenvolvimento do Plan-It, foram utilizadas as seguintes tecnologias principais: Angular para o desenvolvimento da interface com o usuário *(frontend),* Spring Boot para a implementação da lógica de negócio e da API RESTful (backend), e MySQL como sistema de gerenciamento para a persistência dos dados.

A Visão Geral da Arquitetura é apresentada na Figura 2, detalhando a interação entre as camadas implementadas:

Figura 2. Visão geral da arquitetura do sistema, mostrando frontend Angular, Spring Boot API, camada JPA, banco MySQL



Fonte: O AUTOR, 2025

Conforme a Figura 2, o sistema opera em uma estrutura de três camadas, onde:

1. O Frontend (Angular) é a camada de Apresentação, responsável pela interface do usuário e pelas requisições enviadas.

2. O Backend (Spring Boot) atua como a Lógica de Negócio (ou Lógica do Aplicativo). Ele processa as requisições enviadas pelo frontend e utiliza a API RESTful para expor as funcionalidades essenciais (CRUD) sobre as entidades do sistema (Usuário, Projeto, Tarefa e Apontamento).

3. O JPA (Java Persistence API) é a camada de Persistência utilizada pelo Spring Boot para acessar e gerenciar os dados no banco de dados.

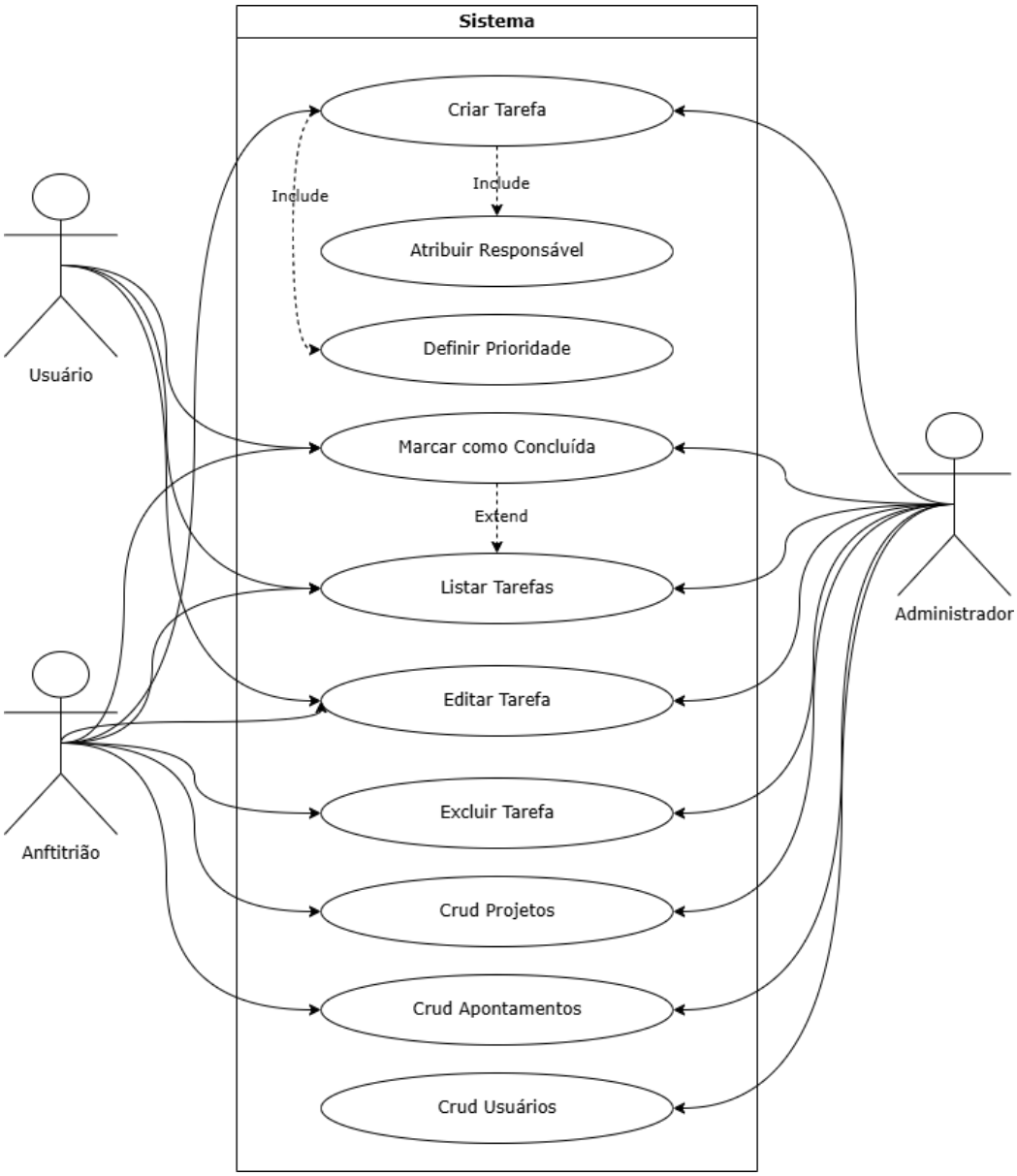
4. O MySQL é o Sistema de Gerenciamento de Banco de Dados (SGBD) que armazena as informações.

A interação típica mostrada na arquitetura ocorre quando "o usuário envia requisições via interface web, que são encaminhadas ao *backend*. O Spring Boot processa a lógica de negócio, persiste ou consulta dados por meio do JPA". Essa separação clara facilita a manutenção e a escalabilidade do Plan-It.

3.1.1. Diagrama de Casos de Uso

O diagrama de casos de uso apresenta os principais atores e funcionalidades do sistema Plan-It, agrupadas em pacotes funcionais. Ele serve para validar o escopo do projeto e garantir que cada requisito seja atendido pela implementação.

Figura 3. Diagrama de casos de uso do sistema de gerenciamento de tarefas, organizado em três pacotes funcionais



Fonte: O Autor, 2025

Conforme apresentado na Figura 3, o Plan-It define dois principais atores, o Usuário/Colaborador e o Administrador, com níveis de privilégio distintos.

O ator Usuário/Colaborador representa o público-alvo principal que busca a centralização da informação e o controle sobre as atividades. Suas funcionalidades essenciais estão concentradas na gestão de atividades e projetos a ele designados. Especificamente, o Usuário/Colaborador possui permissão para:

- Criar, editar e listar tarefas;
- Listar projetos que o envolvam.

O ator Administrador, por sua vez, herda as permissões do Colaborador, mas detém privilégios adicionais para a manutenção e gestão do sistema. Suas responsabilidades abrangem:

• Criação e exclusão de projetos e tarefas;

• Gestão de usuários (incluindo a listagem e o cadastro de novas contas);

• Gerenciamento de parâmetros do sistema.

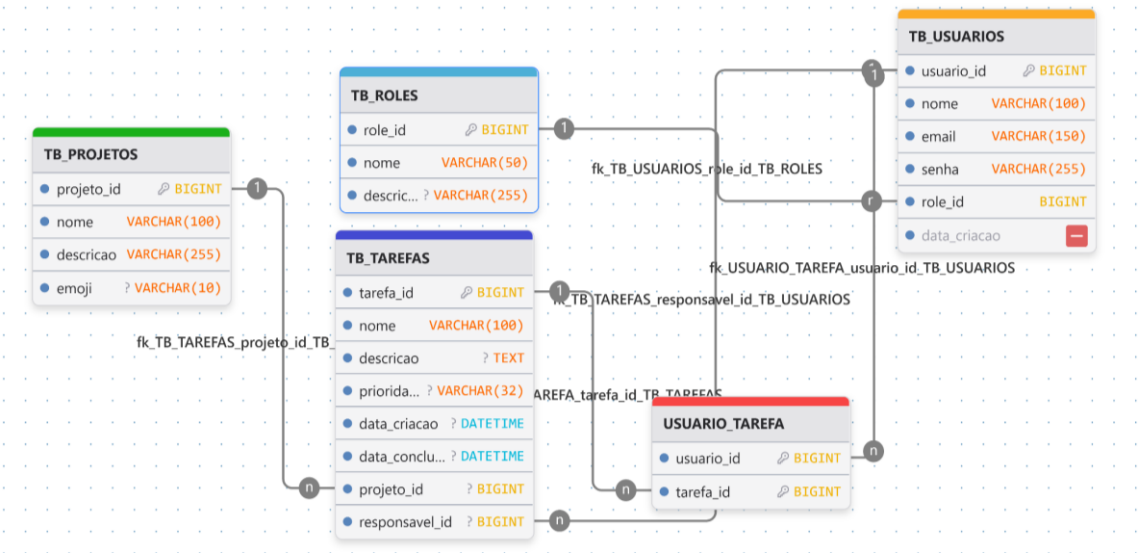
A distinção clara de responsabilidades entre os atores, conforme mapeado no Diagrama de Casos de Uso, suporta a Validação da Regra de Negócio do sistema, garantindo que o acesso às funcionalidades administrativas seja restrito, o que é crucial para a segurança e integridade do modelo de dados.

3.2. Modelagem de Dados

A modelagem de dados define as entidades principais do sistema (TB_USUARIOS, TB_PROJETOS, TB_TAREFAS e TB_ROLES) e seus relacionamentos. A estrutura garante a integridade referencial e reflete as regras de negócio, como demonstrado na Figura 4.

No modelo, cada tarefa (TB_TAREFAS) pertence a um projeto (TB_PROJETOS) e possui um usuário principal designado como responsável (através da chave estrangeira responsavel_id). A tabela TB_ROLES é utilizada para categorizar os usuários, definindo seus perfis de acesso e permissões dentro do sistema.

Figura 4. Diagrama Entidade-Relacionamento do sistema, mostrando as tabelas TB_USUARIOS, TB_PROJETOS, TB_TAREFAS, TB_ROLES, USUARIO_TAREFA e seus relacionamentos.
USUÁRIO, PROJETO e TAREFA e seus relacionamentos



Fonte: O Autor, 2025

No detalhamento do modelo, a tabela TB_USUARIOS armazena as informações dos colaboradores. TB_PROJETOS contém os metadados dos projetos, e TB_TAREFAS reúne os detalhes de cada atividade.

Uma adição crucial ao diagrama é a tabela associativa USUARIO_TAREFA. Ela implementa um relacionamento N:N (muitos-para-muitos) entre usuários e tarefas, permitindo que múltiplos colaboradores sejam atribuídos a uma mesma tarefa, além do responsável principal já indicado na tabela TB_TAREFAS.

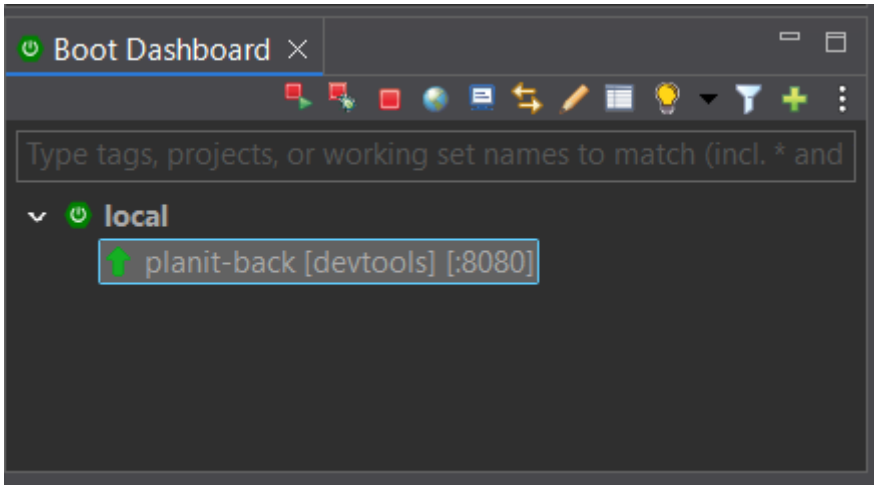
3.3. Configuração de Ambiente

Para o desenvolvimento do *backend* utilizou-se o Spring Tools Suite 4 (STS), uma IDE baseada no Eclipse especialmente otimizada para projetos Spring Boot. O STS fornece autocompletar inteligente para anotações Spring, wizards para geração de código, visualização de contextos de aplicação e ferramentas de depuração avançada. Essas facilidades aceleram o ciclo de desenvolvimento, permitindo criar, executar e inspecionar micro serviços Spring Boot de forma mais ágil e produtiva, como demonstrado pelo painel Boot Dashboard (Figura 6).

Na camada frontend, adotou-se o Visual Studio Code, um editor leve e extensível com amplo suporte à linguagem TypeScript e ao *framework* Angular. Por meio de extensões como Angular Language Service, ESLint e Prettier, o VS Code oferece verificação de sintaxe em tempo real, formatação automática de código e snippets específicos para componentes e serviços. Recursos como o terminal integrado e o controle de versões Git tornam o VS Code uma ferramenta versátil para a construção de interfaces web.

O DBeaver foi utilizado para a gestão do banco de dados, permitindo modelar visualmente o esquema, criar tabelas, executar consultas SQL e gerenciar usuários de forma ágil e segura

Figura 5. Painel Boot Dashboard do STS exibindo a aplicação Plan-It em execução

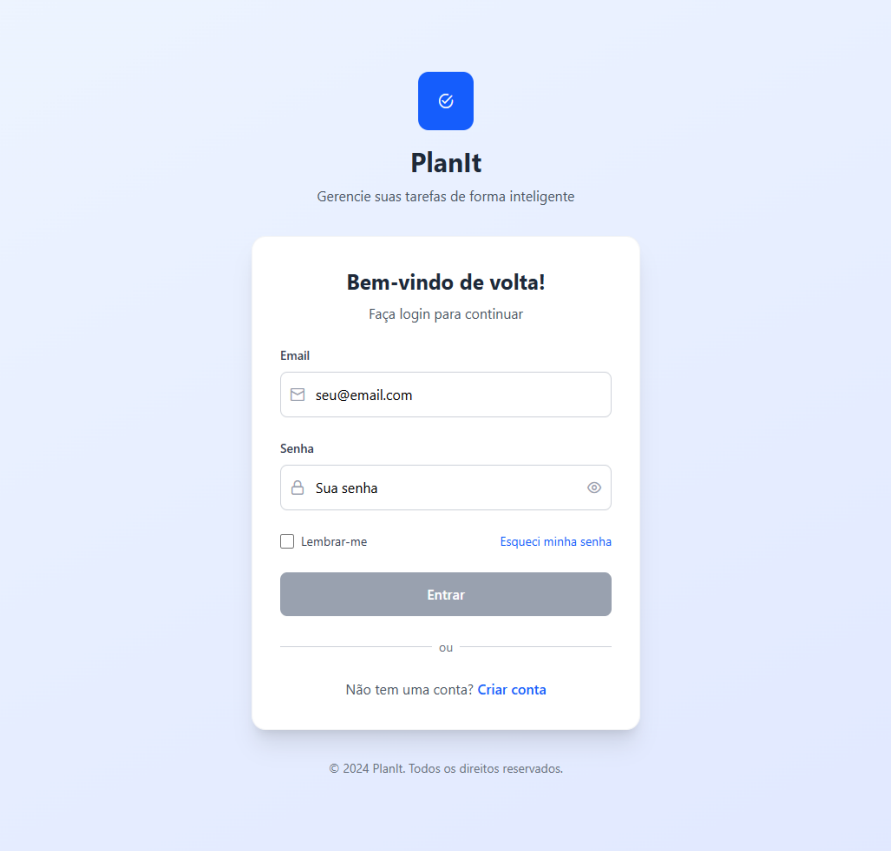


Fonte: O Autor, 2025

3.4. Tela de Login

A funcionalidade de login é o primeiro ponto de interação entre o usuário e o sistema, sendo um componente fundamental para garantir a segurança dos dados e a personalização da experiência do usuário. É responsável por autenticar os usuários cadastrados antes de permitir o acesso às funcionalidades de gerenciamento de tarefas.

Figura 6. Tela de Login do sistema



Fonte: O Autor, 2025

Conforme ilustrado na (Figura 6), a interface solicita as credenciais do usuário e-mail e senha para validação. Ao acionar o botão "Entrar", a aplicação processa a autenticação.

3.5. Tela de Início

Após a autenticação, o usuário é direcionado para a tela inicial, que funciona como o painel de controle central do sistema. Esta interface foi projetada para oferecer uma visão geral e organizada de todas as funções do sistema, facilitando o acesso rápido às páginas e promovendo uma gestão de trabalho eficiente.

Figura 7. Tela de Início do sistema

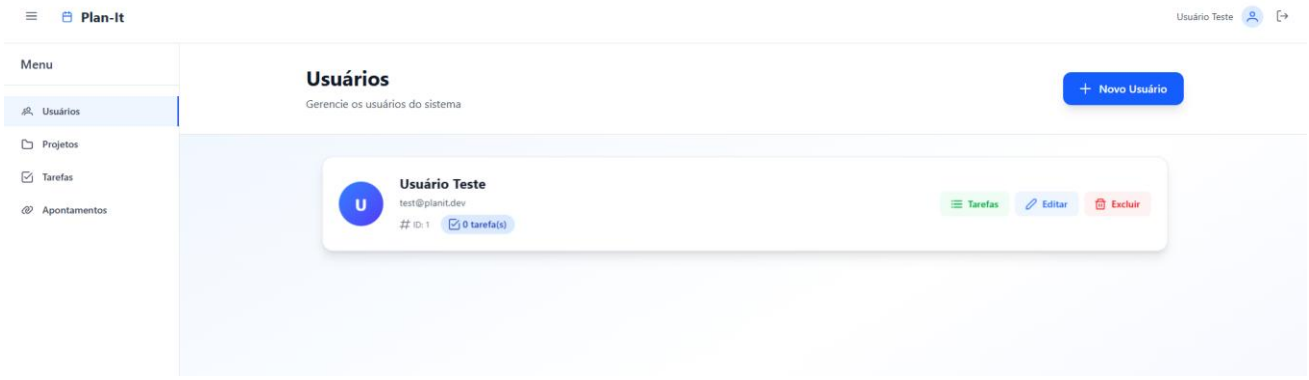
Fonte: O AUTOR, 2025

Conforme (Figura 7), a tela exibe as opções em formato de cartões, apresentando informações essenciais como o título da página e uma breve descrição das funcionalidades delas.

3.6. Tela Listagem de Usuários

A funcionalidade de lista de usuários é uma área administrativa do sistema, projetada para o gerenciamento das contas cadastradas. Esta interface permite que um administrador visualize, adicione e gerencie os perfis de todos os usuários, desempenhando um papel crucial na manutenção e segurança do sistema. O acesso a esta funcionalidade é tipicamente restrito para garantir a integridade dos dados.

Figura 8. Funcionalidade listagem de usuários do sistema



Fonte: O AUTOR, 2025

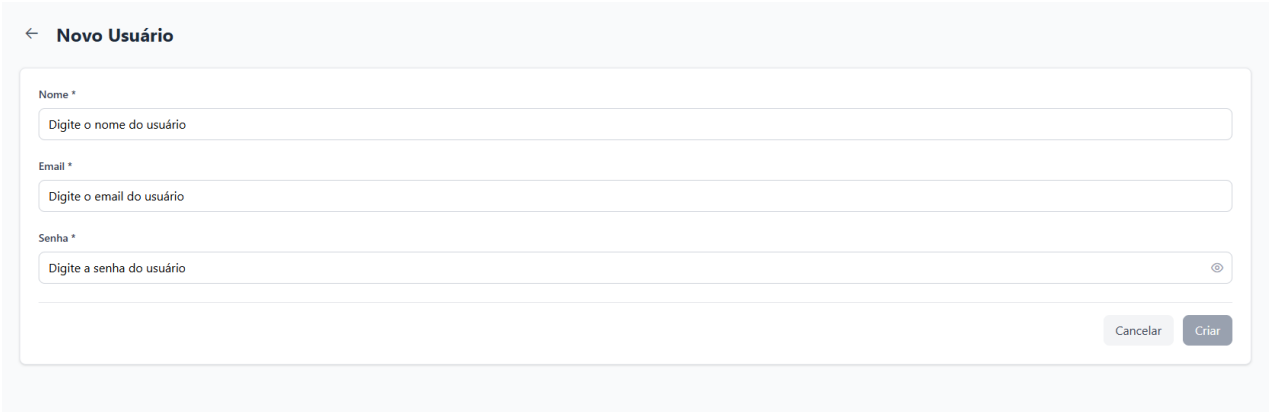
Como a (figura 8) ilustra, a funcionalidade apresenta uma listagem de todos os usuários registrados no sistema, exibindo informações relevantes como nome e e-mail em um formato lista.

A interface inclui funcionalidades essenciais de gerenciamento, como botões para editar ou excluir um usuário existente. Além disso, um botão de destaque, como "+ NOVO USUÁRIO", permite o acesso direto ao formulário de cadastro, agilizando a inclusão de novas contas.

3.7. Tela de Cadastro de Usuários

A funcionalidade de cadastro de usuários é uma parte administrativa do sistema, projetada para permitir que um administrador crie novas contas de usuário diretamente na plataforma. Diferente da tela de cadastro pública, esta interface faz parte do painel de gerenciamento do sistema para oferecer uma experiência de gestão centralizada e segura.

Figura 9. Funcionalidade de cadastro de novos usuários pelo administrador



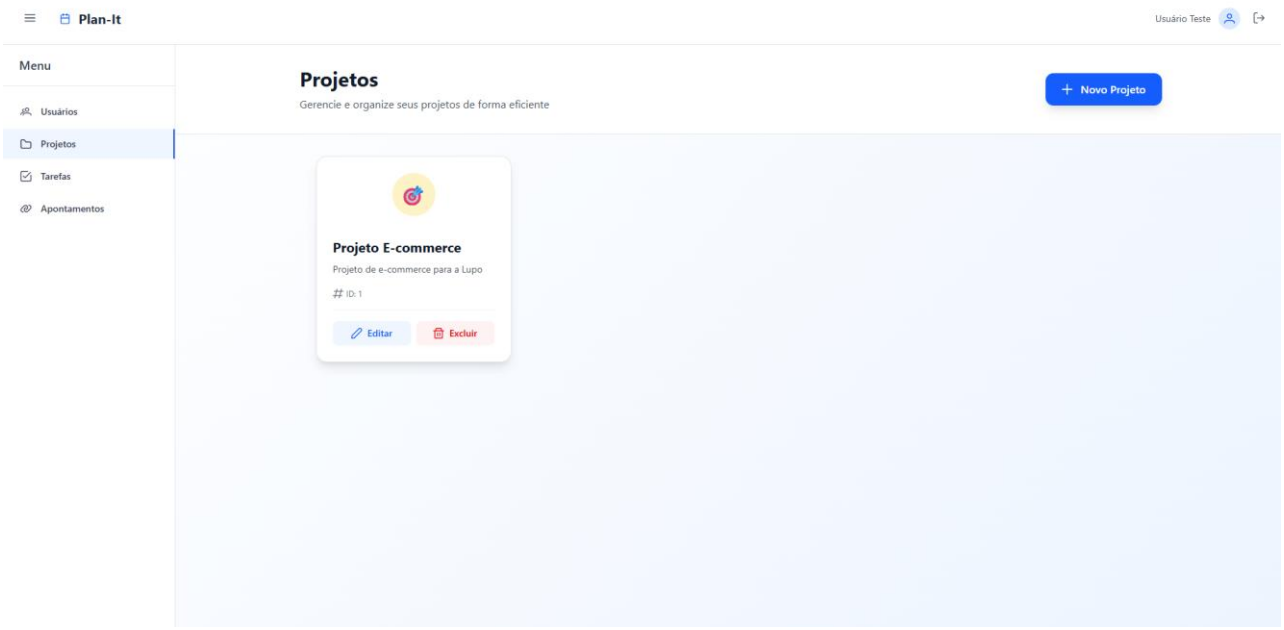
Fonte: O AUTOR, 2025

Esta funcionalidade (Figura 9), apresenta um formulário para a inserção dos dados do novo usuário, como nome, e-mail e uma senha inicial. Ao preencher as informações e acionar o botão "Salvar" ou "Cadastrar", os dados serão persistidos no banco de dados. Esta funcionalidade é complementar à tela de listagem de usuários.

3.8. Tela Listagem de Projetos

A funcionalidade de listagem de projetos é o coração organizacional do sistema "Plan-It". É nesta interface que o usuário obtém uma visão panorâmica de todos os seus projetos em andamento, servindo como um painel de controle para o gerenciamento de suas responsabilidades. Desenvolvida em Angular, esta tela materializa o conceito de ter um sistema centralizado para organizar todas as demandas, um pilar fundamental para a produtividade.

Figura 10. Funcionalidade de listagem de projetos do usuário



Fonte: O Autor, 2025

A interface apresenta todos os projetos cadastrados pelo usuário (Figura 10), exibindo informações cruciais como id, nome, descrição e prazo de conclusão. A tela oferece funcionalidades interativas, como opções para editar ou excluir um projeto. O botão "+ NOVO PROJETO" permanece em destaque, reforçando o ciclo de produtividade e permitindo ao usuário capturar novas demandas assim que surgem.

4. RESULTADOS

Esta seção apresenta e analisa os resultados funcionais e as métricas de qualidade obtidas com a implementação do sistema Plan-It. O desenvolvimento seguiu a arquitetura em três camadas e utilizou Spring Boot no *backend*, validando o cumprimento dos requisitos de gerenciamento de tarefas e a robustez técnica da aplicação.

4.1. Resultados da Implementação Funcional

A implementação do sistema Plan-It comprovou a eficácia da abordagem proposta para o gerenciamento de tarefas, conseguindo centralizar as informações e definir responsabilidades de forma clara.

A API RESTful, desenvolvida em Spring Boot, atuou como a Lógica do Aplicativo, garantindo que todas as operações essenciais (CRUD) sobre as entidades USUARIO, PROJETO e TAREFA fossem executadas de maneira segura e eficiente, com a persistência gerenciada pelo MySQL.

• Validação da Regra de Negócio: O modelo de dados implementado assegurou a integridade referencial, pois as chaves estrangeiras responsavel_id em PROJETO e TAREFA foram aplicadas, garantindo que "todo projeto e toda tarefa estejam vinculados a um usuário existente".

• Centralização da Informação *(Frontend):* A interface do Angular, demonstrada nas telas de listagem e início, resolveu a dificuldade inicial de dispersão de informações, oferecendo uma visão organizada dos projetos em andamento, o que é um pilar fundamental para a produtividade.

• Aceleração da Produtividade: A funcionalidade de listagem de projetos inclui o botão "+ NOVO PROJETO" em destaque, que promove o ciclo de produtividade ao permitir ao usuário "capturar todas as coisas que devem ser feitas [...] num sistema lógico e confiável".

4.2 Métricas de Qualidade e Boas Práticas

Os resultados obtidos na fase de desenvolvimento foram medidos por meio de ferramentas de avaliação contínua, garantindo que o *software* atinja o alto padrão de qualidade necessário para APIs corporativas.

4.2.1. Cobertura de Código (JaCoCo) e Testes Unitários (JUnit)

O uso do JUnit permitiu a criação e execução de testes automáticos para classes e métodos, garantindo que o sistema funcione corretamente e sem falhas, promovendo maior confiança no produto final.

O relatório gerado pelo JaCoCo (Java Code Coverage) forneceu dados quantitativos sobre a robustez do *backend*. A porcentagem de cobertura de código atingida é um indicativo de que a grande maioria das linhas de código foi executada por testes automatizados, o que assegura que as refatorações implementadas "não quebrem funcionalidades existentes". Esta precisão, visualmente destacada pelo relatório, agilizou a identificação de trechos de código sem verificação.

4.2.2. Documentação de API (Swagger)

A documentação da API RESTful do sistema foi implementada através da integração do Swagger ao *backend* Spring Boot. Esta ferramenta automatiza a geração da especificação da API (baseada no padrão OpenAPI) diretamente a partir das anotações presentes no próprio código-fonte. O resultado imediato dessa abordagem é o Swagger UI, uma interface web interativa que fornece uma página visual clara e funcional. Essa interface permite que desenvolvedores, testadores e equipes de frontend possam inspecionar, entender e testar cada *endpoint* em tempo real, o que "simplifica a integração entre sistemas e a comunicação entre desenvolvedores e testadores", fator essencial para a qualidade do *software.*

O Swagger, em essência, é um ecossistema de ferramentas construído em torno da Especificação OpenAPI (OAS). A OAS atua como um "contrato" agnóstico de linguagem para descrever APIs, definindo formalmente seus endpoints, parâmetros de entrada, tipos de resposta, modelos de dados e autenticação. No contexto deste projeto, o Spring Boot gera esse contrato (em formato JSON) e o Swagger UI o consome para renderizar a documentação visual. Manter a documentação sincronizada automaticamente com o código elimina inconsistências e acelera o ciclo de desenvolvimento, garantindo que todos os consumidores da API tenham uma fonte única de verdade.

5. CONSIDERAÇÕES

Este Trabalho de Conclusão de Curso apresentou o desenvolvimento do "Plan-It", um sistema *web* para gerenciamento de tarefas e projetos. O objetivo foi propor uma solução robusta para a dificuldade central observada na introdução: a dispersão de informações, o estresse e a baixa produtividade decorrentes do uso de ferramentas inadequadas, como e-mails e planilhas. A solução foi construída seguindo uma arquitetura em três camadas, utilizando Angular para o *frontend*, Spring Boot para o *backend* e MySQL para a persistência de dados.

As funcionalidades implementadas, como a gestão de projetos, tarefas e usuários (Figuras 8 a 10), foram projetadas para centralizar as informações e otimizar o fluxo de trabalho. A plataforma oferece uma interface organizada que permite a definição clara de responsáveis e prazos, materializando os conceitos de produtividade de David Allen (2015) ao "capturar todas as coisas que devem ser feitas [...] num sistema lógico e confiável".

A qualidade técnica e a robustez do *software* foram aferidas por meio de ferramentas de avaliação contínua. A adoção de testes unitários (JUnit) assegura a confiabilidade das regras de negócio do *backend*. A cobertura de código foi analisada com o JaCoCo para garantir a abrangência dos testes, e a documentação da API com Swagger facilita a manutenção e a integração do sistema

Conclui-se que o "Plan-It" cumpre seu objetivo acadêmico ao entregar uma plataforma de *software* funcional, escalável e tecnicamente validada. O sistema supera as limitações das ferramentas genéricas ao oferecer uma solução dedicada ao gerenciamento de tarefas. A plataforma desenvolvida serve como uma base sólida para futuras expansões, que poderiam incluir testes de usabilidade com o público-alvo, o desenvolvimento de um aplicativo móvel e a integração com outras ferramentas de produtividade.

REFERÊNCIAS

ACELINO, F. Guia prático para construir uma API REST com Spring Boot e Java. **Medium**, 2023. Disponível em: https://medium.com/@felipeacelinoo/guia-pr%C3%A1tico-para-construir-uma-a pi-rest-com-spring-boot-e-java-99fa79f62c7. Acesso em: 24 mar. 2025.

ADRIANO, T. **Guia prático de TypeScript**: Melhores suas aplicações JavaScript. São Paulo: Casa do Código, 2021.

ALLEN, D. **A Arte De Fazer Acontecer**: O método GTD - Getting Things Done: Estratégias para aumentar a produtividade e reduzir o estresse. Tradução de: Afonso Celso da Cunha Serra. Rio de Janeiro: Sextante, 2015.

AZEVEDO, M. Qualidade e JUnit: introduzindo automatização de testes unitários do seu software Java no dia-a-dia. **Medium**, 2018. Disponível em: https://mari-azevedo.medium.com/qualidade-e-junit-introduzindo-automatiza%C3%A7%C3%A3o-de-testes-unit%C3%A1rios-do-seu-software-java-no-dia-a-dia-849611de5574. Acesso em: 07 jun. 2025.

BATISTA, N. Angular: o que é, para que serve e um Guia para iniciar no framework JavaScript. **Alura**, 2025. Disponível em: https://www.alura.com.br/artigos/angular-js?srsltid=AfmBOop5w3FVaElK0-Nitl8XB hkLVHOUPDQjy-9n3-KHqPhRMh6tvVuO. Acesso em: 22 mar. 2025.

ERICKSON, J. MySQL: Entendendo o que é e como é usado. **Oracle**, 2023. Disponível em: https://www.oracle.com/br/mysql/what-is-mysql/. Acesso em: 28 mar. 2025.

FLORENTINO, R. Programação Orientada a Objetos: Polimorfismo. **DEV Community**, 2024. Disponível em: https://dev.to/fabianoflorentino/programacao-orientada-a-objetos-polimorfismo-230b. Acesso em: 29 mar. 2025.

GAMMA, E. et al. **Padrões de Projeto**: Soluções reutilizáveis de software orientado a objetos. Tradução de: Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

GILLIS, A. O que é uma arquitetura de aplicativo de 3 camadas?. **TechTarget**, 2024. Disponível em: https://www.techtarget.com/searchsoftwarequality/definition/3-tier-application. Acesso em: 29 mar. 2025.

MARTINS, A. Documentação de APIs: você conhece o Swagger?. **Medium**, 2022. Disponível em: https://medium.com/inside-picpay/documenta%C3%A7%C3%A3o-de-apis-voc%C3%AA-conhece-o-swagger-fd8b403d27ed. Acesso em: 08 jun. 2025.

MICROSOFT. O que é o Java Spring Boot? Uma introdução ao Spring Boot, a popular ferramenta baseada em Java para desenvolver aplicativos Web e microsserviços. **Microsoft Azure**. Disponível em: https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-java-spring-boot. Acesso em: 24 mar. 2025.

PEREIRA, L. O que é Programação Orientada a Objetos (POO) e para que serve. **Dio**, 2024. Disponível em: https://www.dio.me/articles/o-que-e-programacao-orientada-a-objetos-poo-e-para-que-serve. Acesso em: 29 mar. 2025.

PONTOTEL. Entenda como a gestão de tarefas aumenta a produtividade, melhora o gerenciamento e veja como escolher a melhor ferramenta! **Sankhya**, 2024. Disponível em: https://www.sankhya.com.br/blog/gestao-de-tarefas/. Acesso em: 29 mar. 2025.

PRESSMAN, R; MAXIM, B. **Engenharia De Software**: Uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016. Disponível em: https://archive.org/details/pressman-engenharia-de-software-uma-abordagem-profissional-8a. Acesso em: 22 mar. 2025.

SOMMERVILLE, Ian. Engenharia de Software. 10. ed. Tradução de Luiz Claudio Queiroz. Disponível em: https://archive.org/details/sommerville-engenharia-de-software-10e. Acesso em: 5 out. 2025.

SPRING. **Spring Boot**. 2025. Disponível em: https://spring.io/projects/spring-boot. Acesso em: 05 out. 2025.

VERÍSSIMO, P. Cobertura de Testes com JaCoCo. **Medium**, 2020. Disponível em: https://medium.com/@pedrobverissimo/tutorial-cobertura-de-testes-com-jacoco-833399a2ccbb. Acesso em: 07 jun. 2025.