# Documentação de Projeto

## para o sistema

# GraphTest

**Versão 2.0**

Projeto de sistema elaborado pelo(s) aluno(s) Lucas Cabral Soares e Maria Eduarda Amaral Muniz e apresentado ao curso de **Engenharia de Software** da **PUC Minas** como parte do Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo dos professores Danilo de Quadros Maia Filho, Leonardo Vilela Cardoso, Raphael Ramos Dias Costa, orientação acadêmica do professor Cleiton Silva Tavares e orientação de TCC II do professor (a ser definido no próximo semestre).

**14/12/2025**

# Tabela de Conteúdo

# Histórico de Revisões

| Nome | Data | Razões para Mudança | Versão |
|---|---|---|---|
| Lucas Cabral Soares e Maria Eduarda Amaral Muniz | 28/09/2025 | • Criação do documento;<br>• Adicionando a Seção 1. Introdução;<br>• Adicionando a Seção 2.3. Modelo de Casos de Uso e Histórias de Usuário;<br>• Adicionando a Seção 2.3.1. Diagrama de Casos de Uso;<br>• Adicionando a Seção 2.3.2. Histórias de Usuários<br>• Adicionando a Seção 4. Projeto de Interface com Usuário<br>• Adicionando a Seção 4.1 Esboço das Interfaces Comuns a Todos os Atores | 1.0 |
| Lucas Cabral Soares e Maria Eduarda Amaral | 12/10/2025 | • Criar diagrama de domínio<br>• Criação do diagrama de sequência do sistema<br>• Criação do diagrama de sequência<br>• Criação do diagrama de comunicação (Fluxo GFC) | 1.1 |
| Lucas Cabral Soares | 18/10/2025 | • Atualizar Tabela de Conteúdo adicionando hiperlinks<br>• Ajustar páginas do documento | 1.2 |
| Maria Eduarda Amaral | 01/11/2025 | • Adicionar diagramas de arquitetura e estados | 1.3 |
| Lucas Cabral Soares | 02/11/2025 | • Fazer diagrama de componente<br>• Atualizar diagrama de classes<br>• Atualizar diagrama de caso de uso<br>• Fazer diagrama de Implantação<br>• Fazer diagrama de componentes | 1.4 |

| | | | |
|---|---|---|---|
| Maria Eduarda Amaral | 02/11/2025 | • Adicionar DER e Glossário | 1.4 |
| Lucas Cabral Soares | 15/11/2025 | • Atualização dos diagramas de comunicação<br>• Atualização do diagrama de classe<br>• Atualização do diagrama de caso de uso e histórias de usuário | 1.5 |
| Maria Eduarda Amaral Muniz | 16/11/2025 | • Correções no DER e diagramas de sequência | 1.6 |
| Lucas Cabral Soares e Maria Eduarda Amaral | 17/11/2025 | • Fazer slides de apresentação para pré-banca<br>• Revisão completa no documento do projeto | 1.7 |
| Maria Eduarda Amaral e Lucas Cabral Soares | 18/11/2025 | • Adicionar Cronograma | 1.8 |
| Maria Eduarda Amaral e Lucas Cabral Soares | 29/11/2025 | • Adicionar casos de teste<br>• Adicionar Processo de Implementação | 1.9 |
| Lucas Cabral Soares | 14/12/2025 | • Revisar documento | 2.0 |

# 1. Introdução

Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o Sistema de Geração e Análise de Grafos de Teste. A referência principal para a descrição geral do problema, domínio e requisitos do sistema é o documento de especificação que descreve a visão de domínio do sistema. Tal especificação acompanha este documento. Anexo a este documento também se encontra o Glossário.

O presente documento descreve o projeto do Sistema de Geração e Análise de Grafos de Teste, desenvolvido como parte do Trabalho de Conclusão de Curso (TCC) do curso de Engenharia de Software da Pontifícia Universidade Católica de Minas Gerais (PUC Minas). O sistema tem como objetivo fornecer uma plataforma integrada para a criação, visualização e análise de artefatos de teste de software, abrangendo tanto a perspectiva estrutural (Grafo de Fluxo de Controle – GFC) quanto a funcional (Grafo de Causa e Efeito – GCE).

O sistema foi concebido para atender às necessidades de estudantes, professores e profissionais da área de Tecnologia da Informação (TI), oferecendo um ambiente interativo que permite colar ou importar trechos de código, gerar automaticamente o GFC, aplicar métricas de cobertura estrutural e derivar, a partir de modelos de causa e efeito, tabelas de decisão (TDs) e casos de teste funcionais. A plataforma também possibilita a validação lógica de modelos, o cálculo de

métricas e a exportação de artefatos em diferentes formatos, promovendo o uso educacional e profissional em contextos de ensino, pesquisa e prática de engenharia de software.

Este documento reúne os principais modelos de requisitos, projeto, dados e interface que compõem o sistema. Inclui diagramas de casos de uso, de classes, de sequência e de componentes, bem como os contratos de operações correspondentes, que detalham o comportamento esperado de cada funcionalidade. O conteúdo aqui descrito serve como base técnica para a implementação do sistema, garantindo rastreabilidade entre os requisitos elicitados e os artefatos de projeto gerados.

# 2. Modelos de Usuário e Requisitos

Esta seção tem como finalidade descrever o perfil do usuário e os requisitos funcionais e não funcionais que orientam o desenvolvimento do sistema. Nela são apresentados os atores que interagem com a plataforma, os modelos de usuário desenvolvidos com base em personas e os casos de uso que estruturam o comportamento esperado do sistema.

A seção inicia com a descrição do ator principal, responsável por representar todos os perfis que utilizam o sistema. Em seguida, são apresentados os modelos de usuário, que detalham características, motivações e objetivos por meio de personas representativas. Posteriormente, são descritos o modelo de casos de uso e as histórias de usuário, que expressam as funcionalidades identificadas a partir das necessidades elicitadas durante a análise de requisitos.

## 2.1 Descrição de Atores

Esta seção apresenta os atores que interagem diretamente com o sistema, descrevendo seus papéis, objetivos e principais ações dentro da plataforma. O propósito é identificar quem utiliza o sistema e de que forma essas interações ocorrem, servindo de base para a definição dos casos de uso.

No contexto do Sistema de Geração e Análise de Grafos de Teste, todos os perfis de usuários, como estudantes, professores e profissionais da área, foram unificados sob o ator Usuário, que representa qualquer indivíduo que utiliza a ferramenta para gerar, visualizar e analisar grafos de teste, bem como realizar atividades de cobertura e modelagem de casos de teste.

**Usuário:** representa o indivíduo que interage diretamente com o sistema, abrangendo diferentes perfis, como estudantes, professores, profissionais da área de Engenharia de Software, Testadores e Engenheiros de QA. O usuário utiliza a plataforma para gerar, visualizar e analisar grafos de teste, tanto sob a perspectiva estrutural (Grafo de Fluxo de Controle – GFC) quanto funcional (Grafo de Causa e Efeito – GCE).

Entre suas principais ações estão:

- Importar trechos de código para gerar o GFC;
- Modelar relações de causa e efeito para derivar TDs e casos de teste funcionais;
- Validar modelos e analisar inconsistências;

Esse ator concentra todas as interações externas relevantes do sistema, representando o comportamento de qualquer pessoa que utilize o sistema para fins de ensino, análise ou apoio ao processo de teste de software.

## 2.2 Modelos de Usuários

Esta subseção tem como objetivo descrever os modelos de usuários desenvolvidos por meio da implementação de personas. Para a construção das personas, foram considerados os diferentes perfis que utilizarão a plataforma: estudantes de Engenharia de Software, professores universitários de teste de software e engenheiros de software profissionais.

A Tabela 1 descreve a persona do usuário Julia Alves Silva, uma estudante de Engenharia de Software. Nela é possível perceber que, apesar de interessada no aprendizado, ela encontra dificuldades em compreender os conceitos teóricos de grafos quando apresentados apenas em aula expositiva. Por isso, deseja contar com uma ferramenta interativa que permita visualizar e manipular os grafos de forma prática, associando-os diretamente à derivação de casos de teste.

|  | **Julia Alves Silva** |
|---|---|
| **Descrição** | Julia tem 21 anos, cursa o quinto período de Engenharia de Software em Belo Horizonte e está atualmente matriculada na disciplina de Teste de Software. Apesar de interessada pela área, sente dificuldade em compreender conceitos abstratos de cobertura estrutural apenas a partir de slides. Ela vê na plataforma uma forma de praticar a geração de grafos de fluxo de controle e de causa-efeito, relacionando-os com casos de teste concretos. |
| **Dores** | • Dificuldade em visualizar conceitos abstratos apenas em teoria.<br>• Falta de ferramentas práticas que auxiliem no aprendizado em sala. |

| | |
|---|---|
| **Objetivos** | • Praticar a geração de grafos de fluxo de controle a partir de pequenos trechos de código.<br>• Entender a derivação de casos de teste funcionais por meio de grafos de causa-efeito. |

*Tabela 1. Persona Julia Alves Silva*

A Tabela 2 descreve a persona do usuário Carlos Eduardo Menezes, um engenheiro de software profissional. Sua descrição mostra que ele tem experiência prática com testes, mas frequentemente precisa analisar grandes volumes de código e validar a qualidade dos testes existentes. Ele sente como dor a fragmentação entre ferramentas de cobertura estrutural e funcional, o que dificulta a visão integrada da qualidade de um sistema.

| | |
|---|---|
|  | **Carlos Eduardo** |
| **Descrição** | Carlos tem 34 anos, atua como engenheiro de software em uma empresa de desenvolvimento de sistemas corporativos. Trabalha diariamente com testes unitários e de integração, utilizando ferramentas de cobertura como JaCoCo. No entanto, enfrenta dificuldade para conectar métricas de cobertura estrutural com derivação de testes funcionais, precisando recorrer a diferentes ferramentas e relatórios manuais. |
| **Dores** | • Necessidade de alternar entre ferramentas para avaliar cobertura estrutural e funcional.<br>• Dificuldade em identificar rapidamente lacunas nos testes implementados. |
| **Objetivos** | • Importar código do projeto e gerar automaticamente o grafo de fluxo de controle.<br>• Avaliar cobertura de testes já existentes e identificar pontos não exercitados.<br>• Derivar casos de teste funcionais a partir de requisitos. |

*Tabela 2. Persona Carlos Eduardo*

A Tabela 3 descreve a persona do usuário Mariana Rocha Almeida, professora universitária de Teste de Software. Sua rotina exige o preparo de aulas, exercícios e exemplos práticos. Ela sente

como dor a ausência de ferramentas didáticas que integrem conceitos teóricos e visuais, capazes de apoiar o ensino e a avaliação dos estudantes em disciplinas de teste

|  | **Mariana Rocha Almeida** |
|---|---|
| **Descrição** | Mariana tem 42 anos, é doutora em Ciência da Computação e leciona disciplinas de Qualidade e Teste de Software em uma universidade. Costuma preparar exercícios com base em exemplos de código simples e em especificações de sistemas. Entretanto, a construção manual de grafos e tabelas de decisão é trabalhosa e nem sempre engaja os alunos. |
| **Dores** | • Esforço elevado para criar exemplos manuais de grafos e tabelas de decisão.<br>• Falta de ferramentas que ilustram de forma clara e didática o processo de geração de casos de teste. |
| **Objetivos** | • Utilizar a plataforma em sala de aula para demonstrar a geração de grafos e casos de teste.<br>• Explorar visualmente a relação entre requisitos, fluxos de controle e teste estrutural/funcional. |

*Tabela 3. Persona Mariana Rocha Almeida*

## 2.3 Modelo de Casos de Uso e Histórias de Usuários

Esta subseção tem como objetivo descrever os casos de uso e histórias de usuário previstos para o projeto. Para isso é apresentado um diagrama de casos de uso onde todos são listados. Dessa mesma forma, também são apresentadas as histórias de usuário relacionadas às funcionalidades previstas para o sistema a ser desenvolvido.

### 2.3.1 Diagrama de Casos de Uso

A Figura 1 apresenta o Diagrama de Casos de Uso da plataforma. O diagrama concentra todas as interações em um único ator, o Usuário, que representa qualquer indivíduo que utiliza o ambiente para conduzir as etapas apoiadas pelo sistema. Esse ator é responsável por iniciar o acesso à plataforma e, a partir disso, executar o ciclo completo de atividades relacionadas ao

processamento de código-fonte e à geração de artefatos. Entre essas atividades estão a gestão de projetos, a importação de arquivos Java, a construção de grafos analíticos — como o GFC e o GCE — e a derivação de produtos de teste associados, incluindo assinaturas de teste estrutural e funcional, métricas de complexidade e tabelas de decisão. O diagrama organiza essas funcionalidades como um conjunto de operações que refletem o fluxo natural de uso da plataforma, desde a entrada do Usuário até a obtenção dos resultados finais do processo de análise.

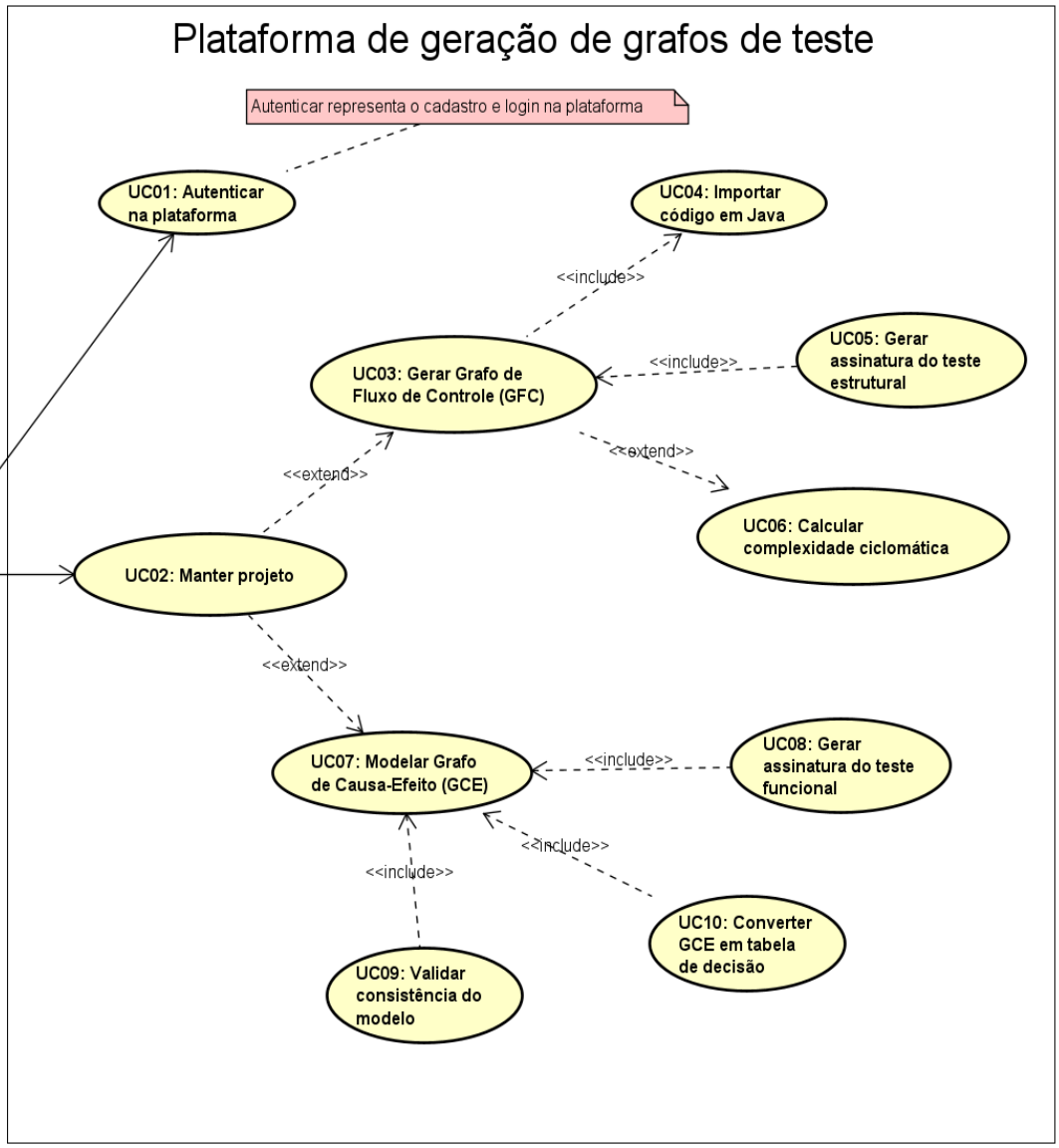


*Figura 1. Diagrama de Caso de Uso*

### 2.3.2 Histórias de Usuário

As histórias de usuários apresentadas nesta seção representam os artefatos que orientam o desenvolvimento do sistema. As histórias de usuários se relacionam com os casos de uso, onde para cada caso de uso tem-se uma história de usuário. Desse modo', relacionam-se da seguinte forma: para cada caso de uso (UC, do inglês *Use Case*) tem-se uma história de usuário (US, do inglês *User Story*), sendo UC01 relacionado ao US01, UC02 relacionado ao US02 e assim por diante. Portanto, as histórias de usuário identificadas para o sistema são:

US01. Como usuário, desejo me cadastrar e me autenticar na plataforma para acessar minhas funcionalidades.

US02. Como usuário, desejo manter meus projetos, incluindo criação, atualização, listagem e remoção.

US03. Como usuário, desejo gerar o GFC de um método para analisar estruturalmente o código.

US04. Como usuário, desejo importar arquivos Java para gerar o GFC.

US05. Como usuário, desejo gerar a assinatura do teste estrutural vinculada ao GFC para facilitar a construção de testes unitários.

US06. Como usuário, desejo calcular a complexidade ciclomática para avaliar o nível estrutural do método analisado.

US07. Como usuário, desejo modelar o GCE para representar relações entre causas e efeitos.

US08. Como usuário, desejo gerar a assinatura do teste funcional derivada do GCE para facilitar a construção de testes unitários.

US09. Como usuário, desejo validar automaticamente a consistência do modelo GCE.

US10. Como usuário, desejo converter o GCE em uma TD para derivar casos de teste funcionais.

## 2.4 Diagrama de Sequência do Sistema e Contrato de Operações

Esta seção apresenta os Diagramas de Sequência do Sistema (DSS), que tem como finalidade representar a interação entre o ator principal e o sistema durante a execução de cada caso de uso. Diferentemente dos diagramas de sequência detalhados com múltiplos objetos, aqui o foco está na visão de alto nível do comportamento do sistema, mostrando como o usuário (ator) comunica-se diretamente com o software por meio de suas principais operações.

Cada Diagrama de Sequência do Sistema (DSS) descreve o fluxo de mensagens entre o ator “Usuário” e o “Sistema DSS”, representando as solicitações realizadas pelo usuário e as respostas processadas pelo sistema. As mensagens são dispostas em ordem cronológica, evidenciando o caminho lógico seguido para a execução de cada funcionalidade. Essa representação permite compreender como o sistema se comporta externamente frente às ações do usuário, sem detalhar a decomposição interna dos módulos.

Associado a cada diagrama, apresenta-se um Contrato de Operações, que define formalmente a operação realizada pelo sistema, incluindo pré-condições, pós-condições, entradas, saídas e exceções. Esses contratos descrevem, de maneira estruturada, as responsabilidades do sistema e os efeitos esperados após a execução de cada operação, garantindo clareza no comportamento funcional e nos limites de atuação do software.

O diagrama da Figura 2 representa a interação entre o ator Usuário e o Sistema DSS durante o processo de autenticação na plataforma, correspondente ao caso de uso UC01 – Autenticar na plataforma. O fluxo tem início quando o usuário insere suas credenciais de acesso (usuário e senha) e solicita a autenticação por meio do comando realizaLogin(). O sistema, então, valida as credenciais fornecidas, verificando sua correspondência com os dados armazenados. A partir dessa verificação, o diagrama apresenta um bloco de decisão (alt) com dois caminhos alternativos. No primeiro, quando as credenciais são válidas, o sistema executa as operações iniciaSessao() e carregaDashboard(), concedendo acesso ao ambiente principal da aplicação. No segundo caminho, quando as credenciais são inválidas, o sistema executa mostrarMensagemDeErro(), informando ao usuário que o processo de login não foi concluído com sucesso.

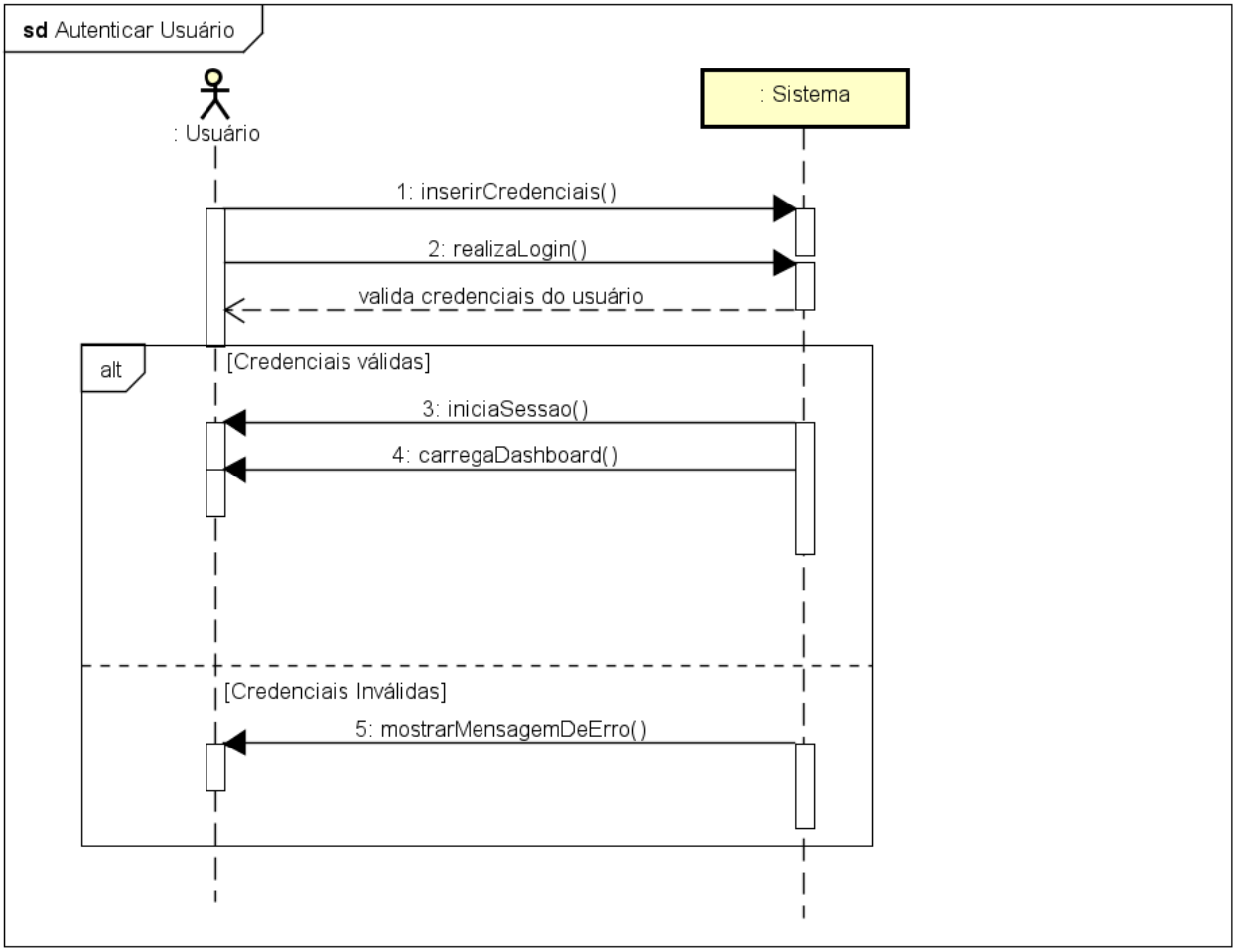


*Figura 2. DSS: Autenticar Usuário*

| **Contrato** | Autenticar Usuário |
|---|---|
| **Operação** | inserirCredenciais() |
| **Referências cruzadas** | UC1: Autenticar na plataforma |
| **Pré-condições** | O usuário precisa estar cadastrado no sistema |
| **Pós-condições** | O usuário precisa estar autenticado |

*Tabela 4. Contrato de Operações: Autenticar Usuário*

O diagrama da Figura 3 representa a interação entre o ator *Usuário* e o *Sistema DSS* no processo de geração do Grafo de Fluxo de Controle (GFC) a partir de um trecho de código importado no editor. O fluxo inicia quando o usuário abre o editor, insere o código e aciona o comando para gerar o GFC. O sistema então realiza etapas internas de normalização do texto, detecção da linguagem e validação de conteúdo. Caso o editor esteja vazio, é exibida uma mensagem de erro e o processo é interrompido. Se o código for válido, o sistema executa o parser para gerar a Árvore Sintática (AST), tratando eventuais erros de análise. Em seguida, o GFC é construído com base na AST, mapeando cada nó e aresta às respectivas linhas do código-fonte. Por fim, o grafo é salvo como artefato e renderizado na interface, sendo apresentado ao usuário com a visualização interativa e o vínculo direto entre o código e a estrutura do grafo.

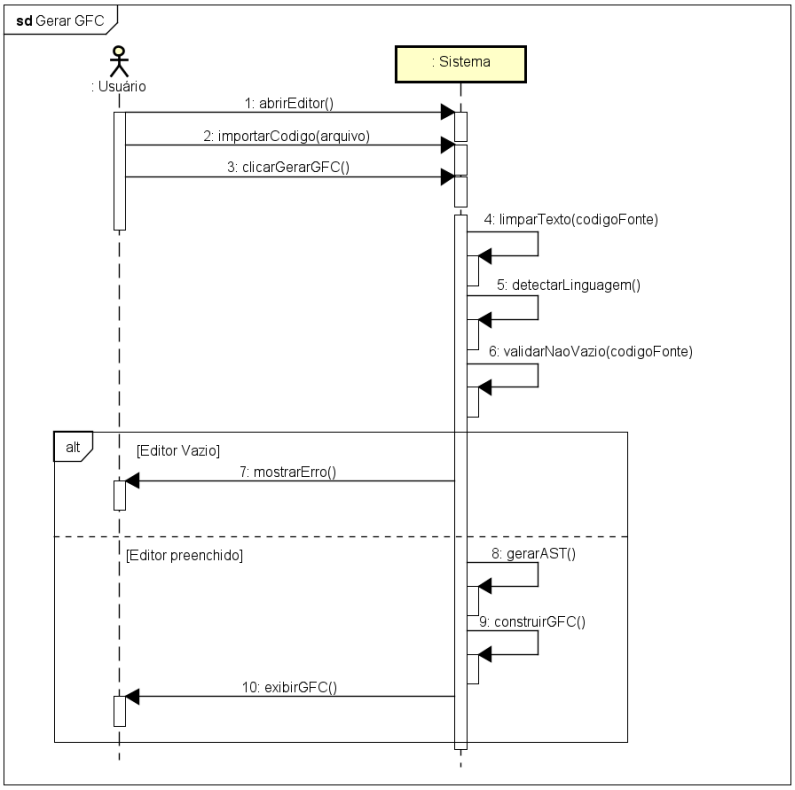


*Figura 3. DSS: Gerar GFC*

| | |
|---|---|
| **Contrato** | Gerar GFC |
| **Operação** | GerarGFC(codigoFonte) |
| **Referências cruzadas** | UC2, UC3, UC4, UC5 |
| **Pré-condições** | O editor precisa estar aberto e o código válido importado |
| **Pós-condições** | GFC gerado, salvo e exibido com mapeamento entre nós e linhas do código |

*Tabela 5. Contrato de Operações: Gerar GFC*

A Figura 4 apresenta a sequência de interações entre o Usuário e o Sistema DSS durante o processo de criação e edição do Grafo de Causa e Efeito (GCE). O fluxo inicia com a abertura do editor funcional, na qual o usuário pode optar por criar um novo grafo ou carregar um existente. Em seguida, o usuário adiciona causas e efeitos, estabelecendo relações entre eles por meio de ligações lógicas. Após definir a estrutura inicial, o usuário aplica operadores lógicos (AND, OR, NOT) e restrições (E, I, O, R, M) para refinar o comportamento do modelo. O sistema atualiza o resumo de regras e re-renderiza o grafo a cada modificação. Quando o usuário solicita a validação, o sistema executa verificações de consistência, detectando causas sem efeito, efeitos inalcançáveis ou conflitos entre restrições. Os resultados são exibidos em um relatório de validação, permitindo correções no modelo. Por fim, o usuário salva o grafo validado, e o sistema confirma a persistência do artefato.

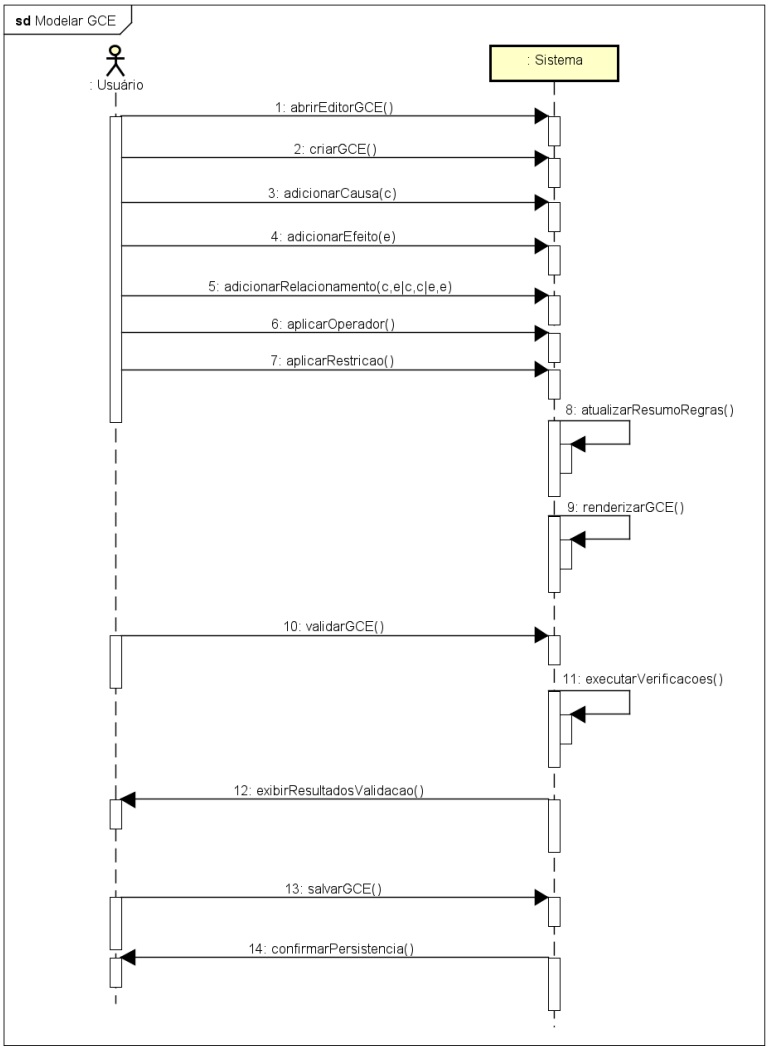


*Figura 4. DSS: Modelar GCE*

| | |
|---|---|
| **Contrato** | GCE (modelagem/operadores/validação) |
| **Operação** | criarGCE(), adicionarCausa(c), adicionarEfeito(e), adicionarRelacionamento(c,elc,cle,e), aplicarOperador(), aplicarRestricao(), validarGCE(), salvarGCE() |
| **Referências cruzadas** | UC11, UC15 |
| **Pré-condições** | O editor GCE precisa estar aberto |
| **Pós-condições** | GCE persistido, regras aplicadas, relatório de validação disponível |

*Tabela 6. Contrato de Operações: Modelar GCE*

A Figura 5 representa a interação entre o Usuário e o Sistema DSS durante o processo de derivação da TD a partir de um Grafo de Causa e Efeito (GCE). O fluxo inicia quando o usuário abre o GCE previamente modelado e seleciona a opção de gerar a TD. O sistema então interpreta o grafo, identificando todas as combinações possíveis de causas e efeitos (mintermos). Em seguida, aplica uma etapa de minimização, eliminando redundâncias lógicas e simplificando as combinações de condições. Após a minimização, o sistema constrói a tabela, organizando as condições (causas) nas colunas e as ações (efeitos) nas linhas correspondentes. Cada linha representa uma regra derivada do GCE, descrevendo os cenários de ativação das ações de acordo com as combinações de causas. Ao final, a TD é persistida no repositório e exibida ao usuário, possibilitando revisão, exportação ou posterior uso na geração de casos de teste funcionais.

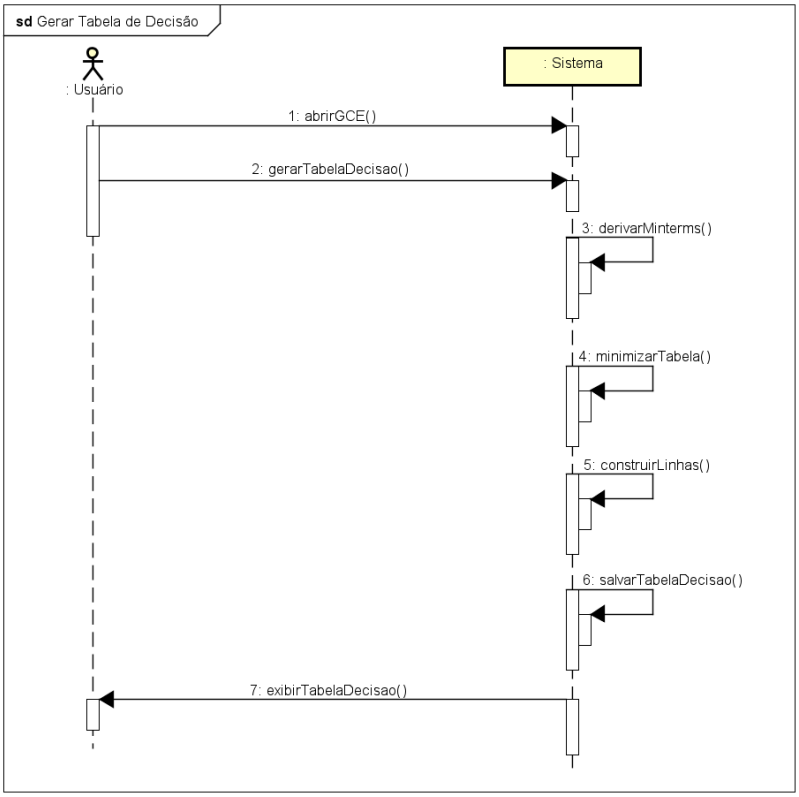


*Figura 5. DSS: Gerar tabela de decisão*

| | |
|---|---|
| **Contrato** | Tabela de Decisão |
| **Operação** | GerarTabelaDecisao() |
| **Referências cruzadas** | UC13 |
| **Pré-condições** | O GCE precisa estar pronto e validado |
| **Pós-condições** | Tabela persistida, traço causas/efeitos → condições/ações |

*Tabela 7. Contrato de Operações: Gerar Tabela de Decisão*

# 3. Modelos de Projeto

Nesta seção são apresentados os modelos de projeto por meio de diagramas na UML. A seção é composta pelos Diagramas de Classes, apresentados na Seção 3.1, os Diagramas de Sequência, apresentados na Seção 3.2, os Diagramas de Comunicação na Seção 3.3, a Arquitetura apresentada na Seção 3.4, os Diagramas de Estado, apresentados na Seção 3.5 e os Diagramas de Componentes e Implantação na Seção 3.6.

## 3.1 Diagrama de Classe

Nesta seção, é apresentado o diagrama de modelo de domínio, o qual representa o modelo lógico do sistema, abrangendo suas entidades, atributos, operações e relacionamentos essenciais. Esse diagrama descreve os objetos do domínio e suas inter-relações, utilizados no tratamento das informações da plataforma. O objetivo desta seção é apresentar os modelos de dados implementados, fornecendo uma visão geral da arquitetura das classes que compõem o sistema e garantindo a consistência e o correto funcionamento da plataforma.

A Figura 6 apresenta o diagrama de classes de modelo de domínio do sistema GraphTest, o qual contempla as principais classes *Usuario, Projeto, ArquivoJava, ClasseJava, MetodoJava, AssinaturaTeste, GrafoDeFluxoDeControle, GFCNo, GFCAresta, GrafoCausaEfeito, GCENo, GCEAresta, Restricao, TabelaDeDecisao, Condicao, Acao e Regra*, responsáveis por armazenar, processar e relacionar os dados manipulados pela plataforma. O modelo inclui classes de associação, como *gcearesta_conecta_gceno* e *gfcaresta_conecta_gfcno*, além das derivadas de associações reflexivas, *gceno_possui_gceno* e *gfcno_possui_gfcno*, empregadas no tratamento de relacionamentos muitos-para-muitos entre entidades estruturais. As classes do núcleo (Usuario e Projeto) representam a organização lógica dos projetos do usuário e suas dependências. As classes que diz respeito ao Teste Estrutural *(ArquivoJava, ClasseJava, MetodoJava, AST, GrafoDeFluxoDeControle, GFCNo e GFCAresta*) são responsáveis pela análise de código-fonte Java e pela geração dos grafos de fluxo de controle. Já as classes que diz respeito ao Teste Funcional (*GrafoCausaEfeito*, *GCENo*, *GCEAresta*, *Restricao*, *TabelaDeDecisao*, *Condicao*, *Acao* e *Regra*) representam os elementos de modelagem do grafo de causa e efeito e da TD, possibilitando a criação e validação de cenários de teste funcional. Todas essas classes são tratadas internamente pelo sistema GraphTest, permitindo o gerenciamento dos projetos, a execução de análises estruturais e funcionais e a consolidação das informações exibidas nas interfaces da plataforma.

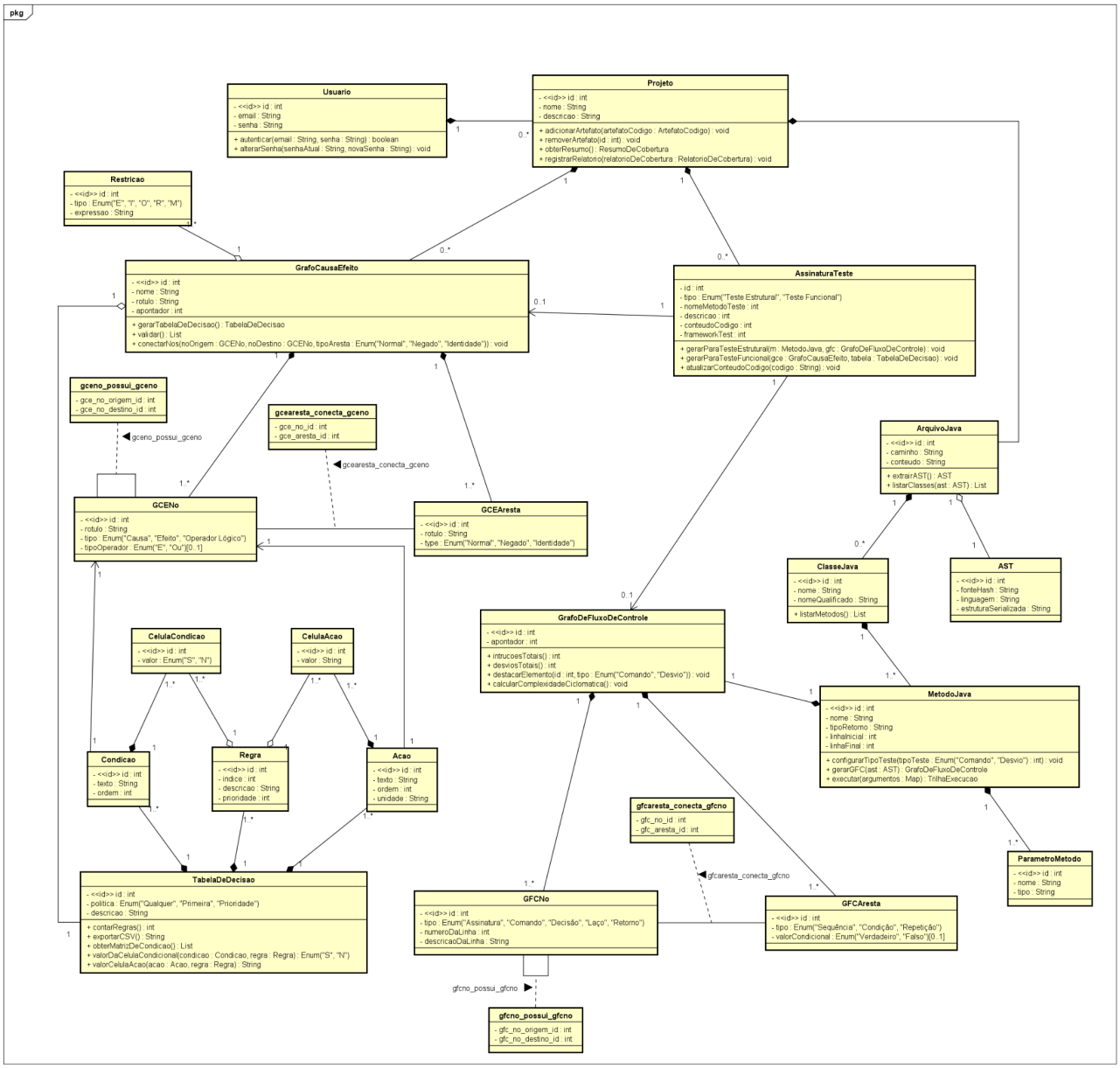


*Figura 6. Diagrama de Classe*

## 3.2 Diagramas de Sequência

Esta seção apresenta os Diagramas de Sequência, que descrevem o comportamento dinâmico do sistema, representando a ordem temporal das interações entre os atores externos, as fronteiras (interfaces), os controles (módulos de processamento) e as entidades (dados persistidos). Cada diagrama ilustra como as mensagens são trocadas entre esses elementos para realizar um determinado fluxo funcional, evidenciando o encadeamento lógico das operações dentro do sistema.

No contexto deste projeto, os diagramas de sequência foram elaborados de forma a detalhar os principais fluxos de execução das funcionalidades da plataforma, contemplando tanto as ações estruturais (relacionadas ao Grafo de Fluxo de Controle – GFC) quanto as funcionais (relacionadas ao Grafo de Causa e Efeito – GCE), além dos processos complementares, como a geração de tabelas de decisão.

Cada diagrama apresenta os módulos internos do sistema (como editores, motores de análise e repositórios) e suas interações com serviços externos, tais como conectores de repositórios de código, validadores lógicos e bibliotecas de minimização. Também são destacadas as fronteiras de comunicação entre o software e os atores externos, mostrando claramente onde as mensagens ultrapassam os limites do sistema.

A Figura 7 representa as interações entre o Usuário, as fronteiras externas e os módulos internos do sistema no processo de geração, análise e visualização do Grafo de Fluxo de Controle (GFC). Ele cobre os casos de uso UC2 – Fornecer código fonte, UC4 – Importar código de repositório, UC10 – Calcular métricas de cobertura estrutural, UC9 – Relatório de elementos não cobertos, UC8 – Mapear casos de teste aos elementos do GFC e UC5 – Gerar Grafo de Fluxo de Controle.

O fluxo inicia quando o Usuário acessa o módulo de GFC por meio da interface web (UI/WebClient). Nesse momento, o usuário pode importar arquivos java. Nesse caso, o código é processado pelo EditorDeCodigo e encaminhado ao módulo AnaliseDeCodigo, responsável por gerar a Árvore Sintática Abstrata (AST).

A geração da AST é realizada por um serviço externo (ParserService), configurado como uma fronteira de integração. O resultado dessa análise é devolvido ao sistema, que então constrói o GFC e o persiste no RepositorioCFG(DB). Em seguida, o grafo é renderizado por meio do motor gráfico externo (RenderizadorGraficos), sendo exibido visualmente na interface para o usuário (UC5).

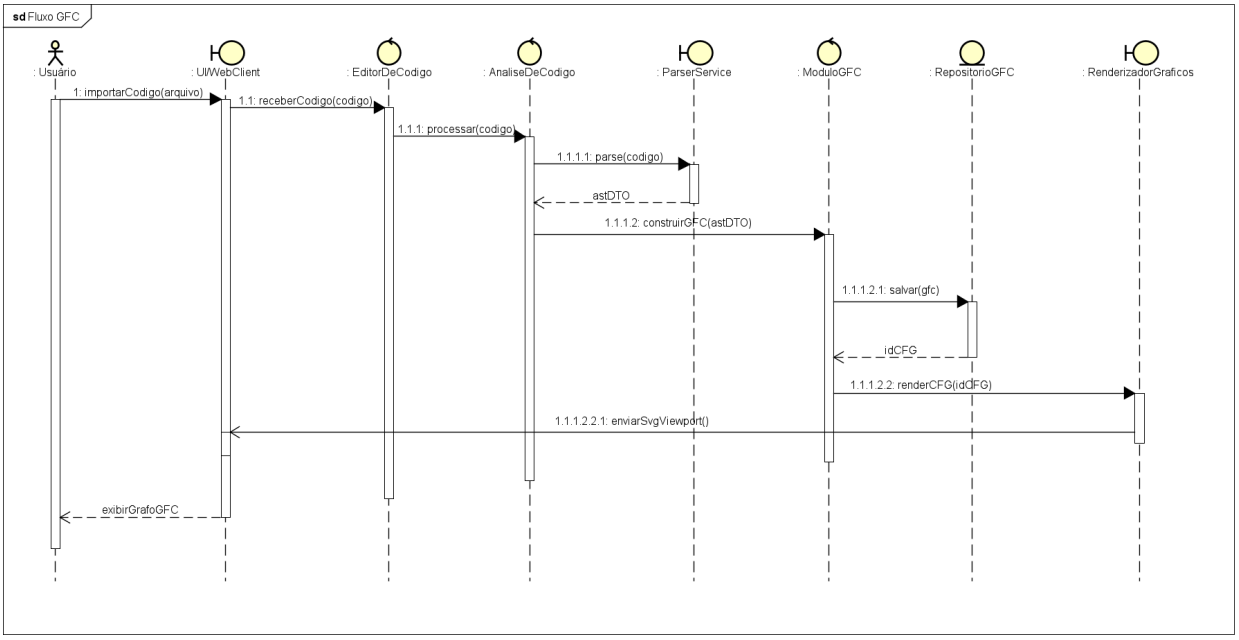

*Figura 7. Diagrama de Sequência: Fluxo GFC*

A Figura 8 apresenta as interações entre o Usuário, as fronteiras externas e os módulos internos do sistema durante o processo de modelagem, aplicação de operadores e validação do Grafo de Causa e Efeito (GCE). Ele cobre os casos de uso UC11 – Modelar GCE e UC15 – Validar consistência do GCE.

O fluxo inicia quando o Usuário acessa o editor de GCE por meio da interface web (UI/WebClient), podendo criar um novo modelo ou abrir um grafo já existente. Dentro do editor, o usuário adiciona causas e efeitos, estabelece conexões entre eles e aplica operadores lógicos (AND, OR, NOT) e restrições do tipo E, I, O, R e M. Essas interações são processadas pelo EditorGCE, que mantém o estado do grafo e envia as alterações ao MotorDeRegra responsável por recalcular as dependências e atualizar o resumo lógico apresentado na interface.

Quando o usuário solicita a validação do modelo, o UI/WebClient encaminha o grafo ao serviço externo Solver/Validador(API), representado como uma fronteira de integração. Esse serviço executa verificações de consistência para detectar erros como causas não utilizadas, efeitos inalcançáveis e conflitos de restrições. O relatório de validação retornado é repassado ao EditorGCE, que o formata e envia de volta à interface, permitindo que o usuário visualize as inconsistências diretamente no grafo e realize ajustes no modelo.

Por fim, o EditorGCE salva o estado atualizado do grafo no RepositorioArtefatos(DB), assegurando que as modificações e resultados de validação fiquem armazenados para uso posterior.

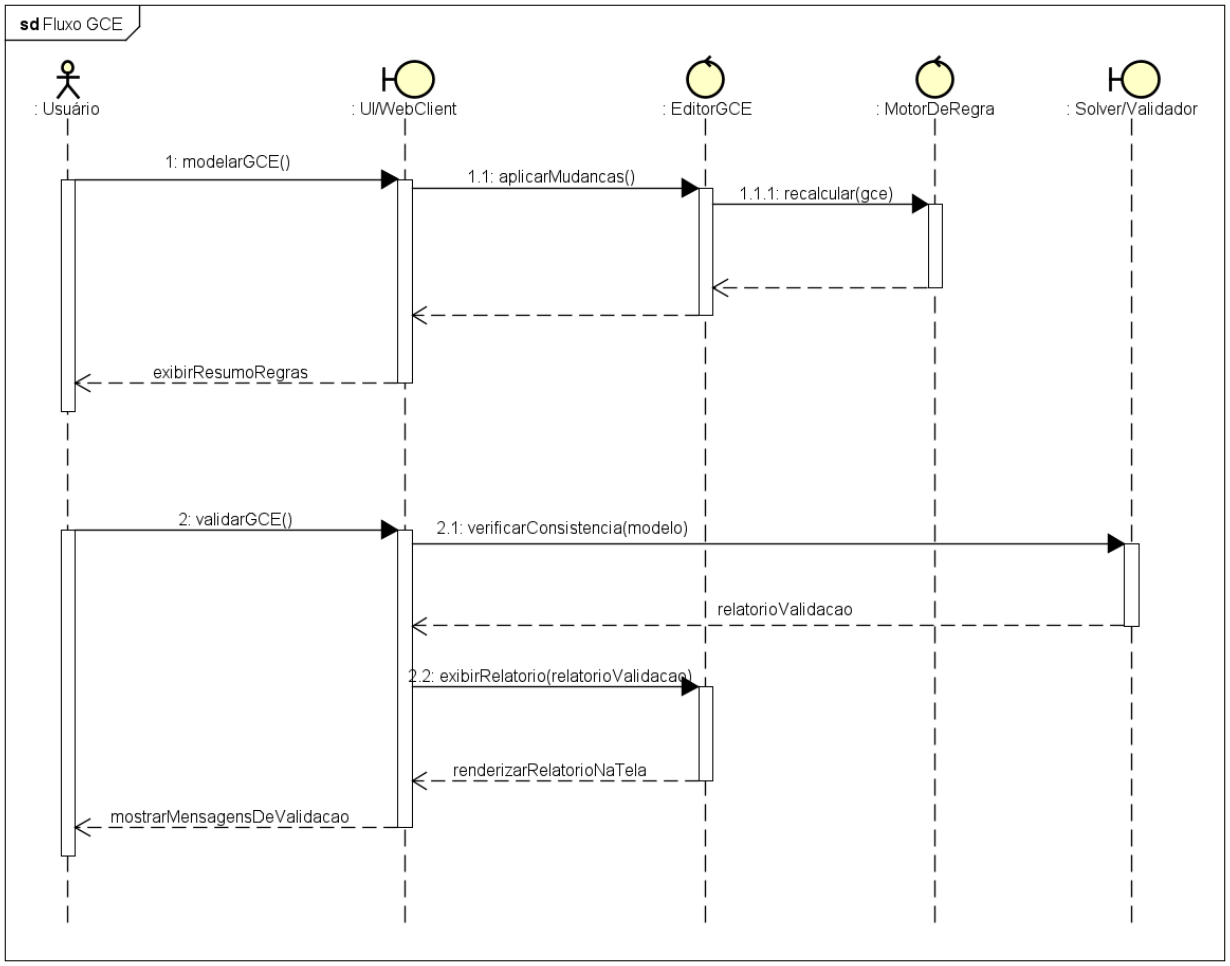


*Figura 8. Diagrama de Sequência: Fluxo GCE*

A Figura 9 descreve as interações entre o Usuário, as fronteiras externas e os módulos internos do sistema durante as etapas de derivação da Tabela de Decisão a partir do GCE. Ele cobre o caso de uso UC13 – Converter GCE em Tabela de Decisão.

O fluxo inicia quando o Usuário, após modelar e validar o GCE, solicita a geração da Tabela de Decisão por meio da interface (UI/WebClient). O módulo DerivadorTabela interpreta o grafo, identifica as combinações lógicas possíveis de causas e efeitos (mintermos) e envia esses dados para a biblioteca externa MinimizationLib(API) — uma fronteira responsável por realizar a minimização lógica das combinações, removendo redundâncias e simplificando o conjunto de regras. O resultado é devolvido ao DerivadorTabela, que constrói a tabela final a partir do conjunto minimizado e a salva no RepositorioArtefatos(DB). Em seguida, a tabela é carregada e exibida ao usuário na interface, permitindo sua análise e edição visual.

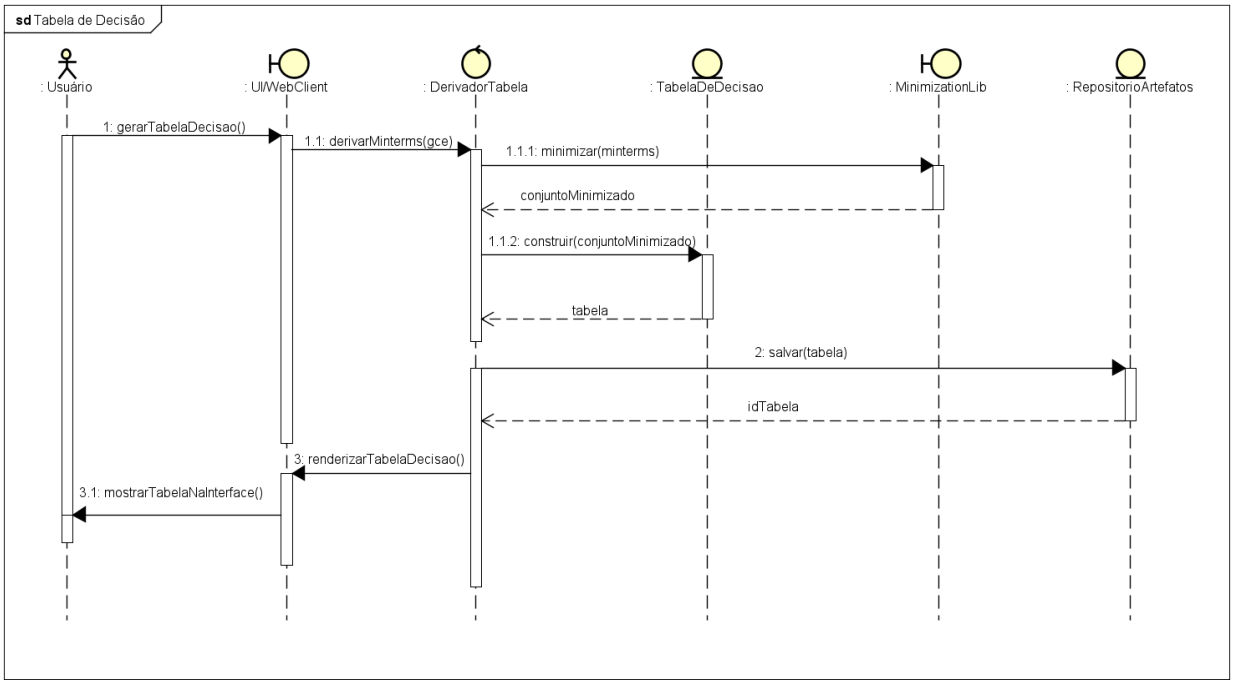


*Figura 9. Diagrama de Sequência: Tabela de Decisão*

## 3.3 Diagramas de Comunicação

Nesta seção são apresentados os diagramas de comunicação que modelam as trocas de mensagens dos principais casos de uso do sistema. A elaboração desses diagramas segue dois níveis distintos de abstração, escolhidos conforme a natureza da funcionalidade representada. Nos diagramas *Dcom01: Autenticar na plataforma* (Figura 10) e *Dcom02: Manter projeto* (Figura 11), adotou-se uma abordagem orientada às camadas da aplicação, evidenciando componentes como *controller*, *service* e *repository*, pois essas funcionalidades dependem diretamente do fluxo técnico da arquitetura *web*. Já os diagramas *Dcom03: Gerar GFC* (Figura 12), *Dcom04: Modelar GCE* (Figura 13) e *Dcom05: Gerar tabela de decisão* (Figura 14) foram modelados com foco no modelo de domínio, utilizando exclusivamente entidades como *Projeto*, *ArquivoJava*, *AST*, *ClasseJava, MetodoJava*, *GrafoDeFluxoDeControle*, *GrafoCausaEfeito* e *TabelaDeDecisao*, uma vez que esses processos representam o núcleo lógico do GraphTest e não se beneficiam da exposição das camadas técnicas. Assim, cada diagrama retrata apenas as interações relevantes ao seu respectivo caso de uso, mantendo coerência entre o nível de abstração e o propósito da funcionalidade.

A Figura 10 apresenta o diagrama de comunicação correspondente ao caso de uso UC01. O processo inicia quando o usuário interage com o *UIWebClient* para enviar seus dados de registro ou credenciais de login. O *AuthController* recebe a solicitação e delega ao *UsuarioService* a criação do novo usuário ou a validação das credenciais informadas. Durante o cadastro, o serviço consulta o repositório, codifica a senha por meio do *PasswordEncoder* e persiste o novo registro. Na autenticação, o serviço compara a senha fornecida com a senha criptografada armazenada.

Com as credenciais validadas, o *AuthController* cria a autenticação, atualiza o contexto de segurança via *SecurityContextHolder* e solicita ao *TokenProvider* a geração do token de acesso. Esse diagrama contempla o caso de uso UC01.

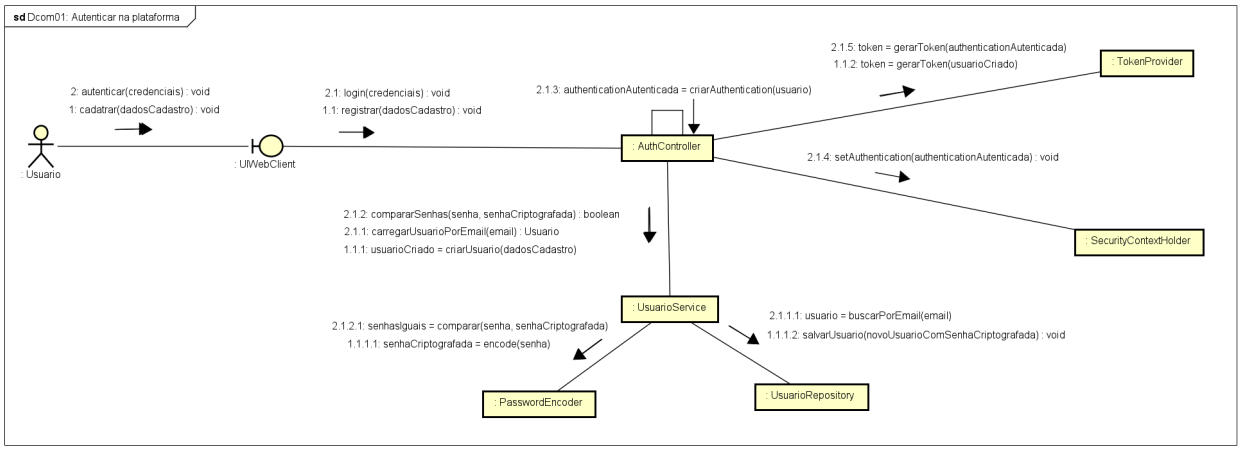


*Figura 10. DCom01: Autenticar na plataforma*

A Figura 11 apresenta o diagrama de comunicação correspondente ao caso de uso UC02. O fluxo inicia quando o usuário interage com o *UIWebClient* para criar, buscar, listar, atualizar ou deletar um projeto. As solicitações são encaminhadas ao *ProjetoController*, que repassa a responsabilidade ao *ProjetoService*. Dentro do serviço, ocorre a lógica de negócio necessária para cada operação, envolvendo a criação de novas instâncias de Projeto, a consulta de projetos existentes e a persistência de alterações. O *ProjetoService* comunica-se com o *ProjetoRepository* para realizar buscas e gravações no armazenamento. Assim, o fluxo descreve de forma integrada como o sistema gerencia todo o ciclo de vida de um projeto, desde sua criação até sua exclusão.

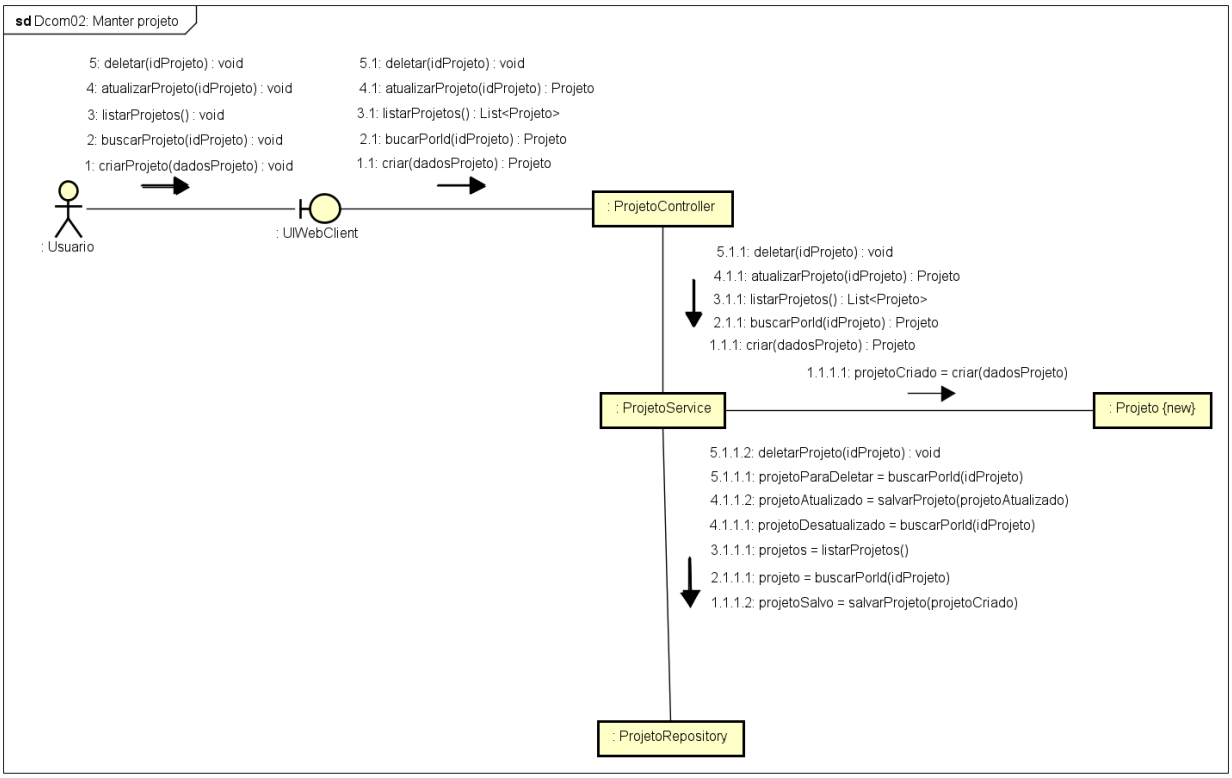


*Figura 11. DCom02: Manter projeto*

A Figura 12 apresenta o diagrama de comunicação que integra os fluxos dos casos de uso UC03, UC04, UC05 e UC06, relacionados à importação do código *Java*, seleção de classes e métodos, geração do GFC, criação da assinatura do teste estrutural e cálculo da complexidade ciclomática. O processo inicia quando o usuário importa um arquivo *Java* pelo *UIWebClien*t, acionando o *Projeto*, que cria e vincula o arquivo ao projeto. O *ArquivoJava* extrai ou recupera o *AST*, organiza as classes e métodos e os vincula às estruturas internas. Em seguida, o usuário seleciona classes e métodos, permitindo que *ClasseJava* e *MetodoJava* criem instâncias representando elementos do código e associem parâmetros com *ParametroMetodo*. Quando o método é selecionado, o sistema gera o GFC por meio das operações de análise do *AST*, criação de nós e arestas e montagem do grafo completo. A partir do GFC, o sistema calcula a complexidade ciclomática e disponibiliza esses dados ao fluxo. Posteriormente, ao requisitar a assinatura do teste estrutural, o sistema utiliza o método atual, o GFC gerado e a estrutura *AST* para produzir uma assinatura contendo o esqueleto do teste e o conteúdo estruturado necessário.

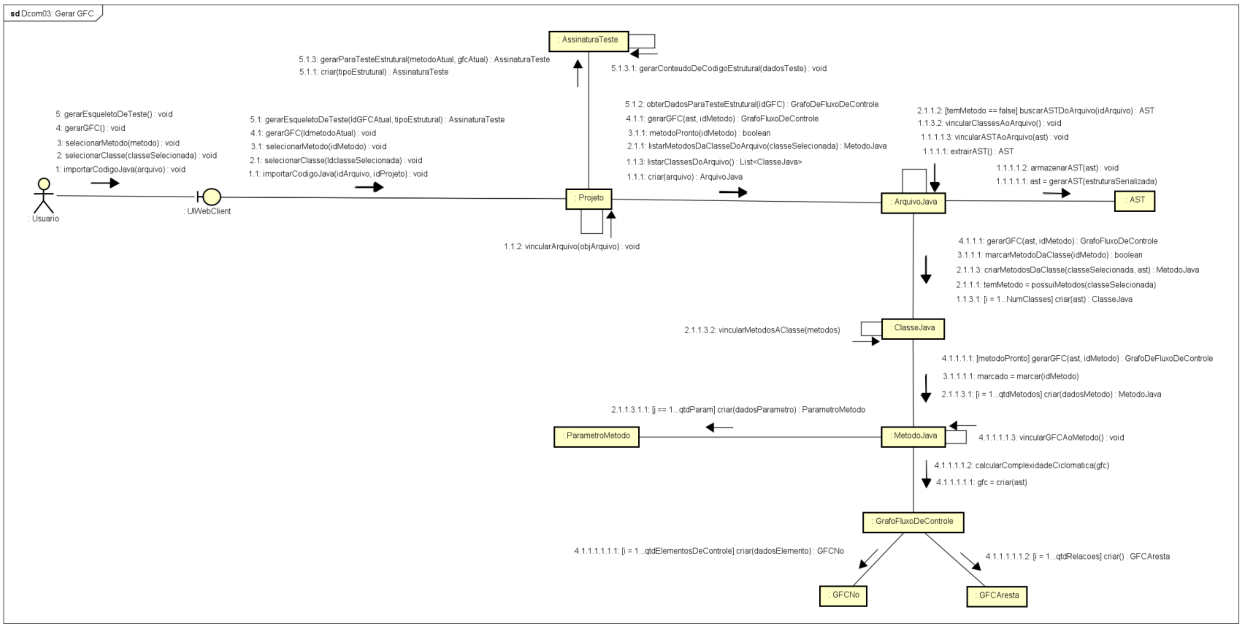


*Figura 12. DCom03: Gerar GFC*

A Figura 13 apresenta o diagrama de comunicação que integra os fluxos dos casos de uso UC07, UC08 e UC09, relacionados à modelagem do Grafo de Causa-Efeito (GCE), aplicação de restrições, verificação das conexões do modelo e geração da assinatura do teste funcional. O processo se inicia quando o usuário aciona a modelagem do GCE por meio do *UIWebClient*, que delega ao *Projeto* a criação de uma nova instância de *GrafoCausaEfeito*. A partir desse ponto, o usuário pode adicionar nós ao grafo, criando elementos dos tipos “CAUSA”, “EFEITO” ou “OPERADOR_LOGICO”, que são instanciados como *GCENo* e incorporados à estrutura do GCE. Em seguida, o usuário pode conectar esses nós, operação que envolve validações internas do grafo e resulta na criação de objetos *GCEAresta*, representando relações adequadas entre os elementos do modelo. O usuário também pode aplicar restrições sobre conjuntos de nós, momento em que o GCE consulta seus elementos, valida a consistência das combinações e cria instâncias de *Restricao*, que passam a compor o modelo. Uma vez concluída a modelagem e validação do GCE, o usuário solicita a geração da assinatura do teste funcional; o *Projeto*, então, aciona a entidade *AssinaturaTeste*, que utiliza o grafo construído para extrair os dados funcionais relevantes, gerar o conteúdo da assinatura e fornecer o esqueleto do teste.

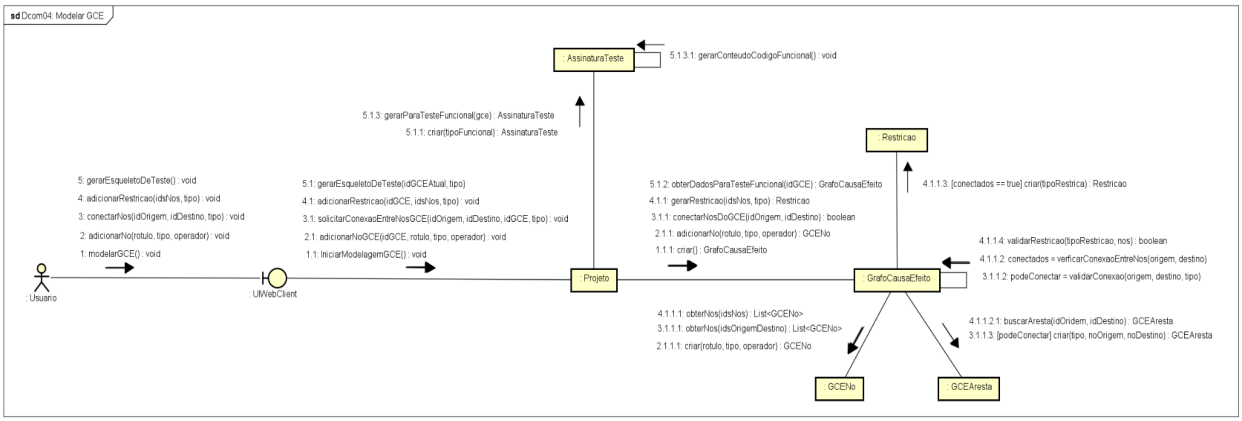


*Figura 13. DCom04: Modelar GCE*

A Figura 14 apresenta o diagrama de comunicação que integra o fluxo do caso de uso UC10, responsável pela geração da tabela de decisão a partir de um GCE previamente modelado e validado. O processo se inicia quando o usuário solicita a criação da tabela por meio do *UIWebClient*, que encaminha a requisição ao *Projeto*. A partir disso, o *Projeto* aciona o *GrafoCausaEfeito*, que disponibiliza suas estruturas internas — incluindo causas, efeitos e regras derivadas — necessárias para compor a tabela. Com essas informações, o *Projeto* coordena a criação da *TabelaDeDecisao*, instanciando inicialmente as condições e ações conforme os nós de causa e efeito definidos no GCE. Na sequência, são criadas as regras, estruturadas de acordo com as combinações lógicas derivadas do modelo, e preenchidas as células condicionais e de ação que compõem a matriz final, representadas pelas instâncias de *CelulaCondicao* e *CelulaAcao*. Esse fluxo descreve a interação organizada entre *Projeto*, *GrafoCausaEfeito* e *TabelaDeDecisao* para transformar o modelo de causa-efeito em uma representação tabular.

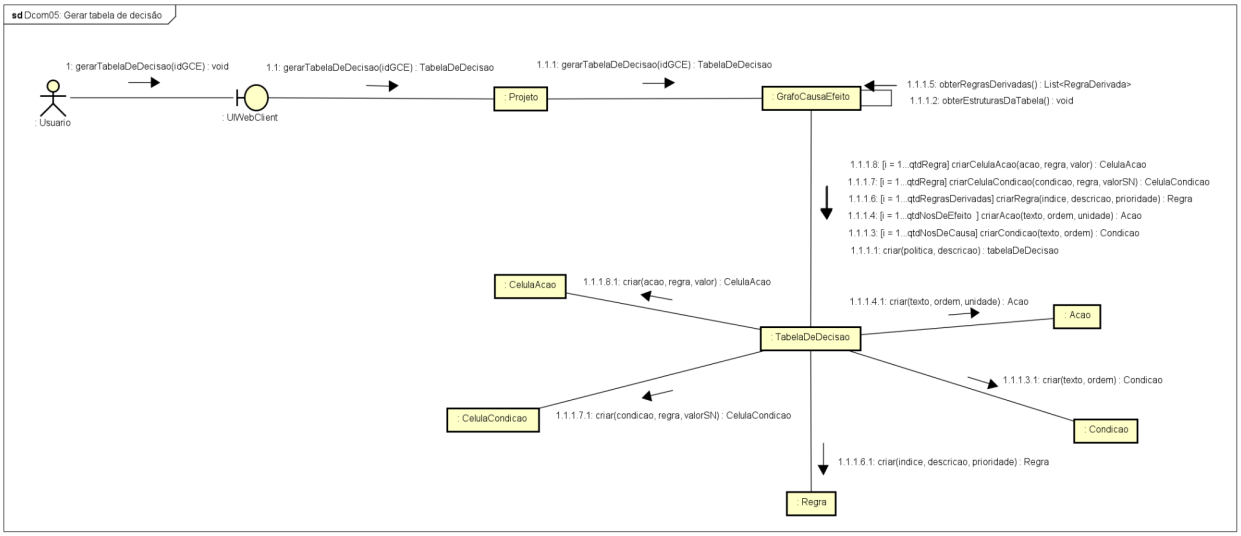


*Figura 14. DCom05: Gerar tabela de decisão*

## 3.4 Arquitetura

A Figura 15 representa a organização estrutural dos principais pacotes e subsistemas da aplicação, evidenciando as camadas de responsabilidade e as dependências entre elas. A arquitetura segue um modelo modular em camadas, que favorece a separação de preocupações, a reutilização de componentes e a manutenção evolutiva do sistema.

Na camada superior, o subsistema UI (*User Interface*) compreende os elementos responsáveis pela interação com o usuário, incluindo as *views* e components que realizam a comunicação direta com o usuário final. Essa camada se comunica unicamente com o subsistema AppGateway, que atua como a fronteira entre a interface e o núcleo da aplicação.

O subsistema AppGateway centraliza os pontos de entrada do sistema, sendo responsável pelo controle de acesso, roteamento de requisições e gerenciamento de propriedades globais. Ele contém módulos como Controllers, que expõem os serviços ao front-end; Auth, responsável por autenticação e autorização; e Properties, que mantém configurações de ambiente e parâmetros de execução.

A camada intermediária Core implementa as regras de negócio do GraphTest. Ela é composta por dois módulos principais: Services, que encapsulam a lógica funcional da aplicação (como geração de grafos, cálculos de métricas e manipulação de artefatos), e Helpers, que fornecem funções auxiliares e utilitárias para suporte às operações dos serviços e controladores. O Core é o coração da aplicação e mantém dependência direta das camadas Domain e Infrastructure.

O subsistema Domain modela os conceitos centrais do domínio da aplicação, assegurando a consistência semântica dos artefatos manipulados. Ele é composto pelos pacotes Models, DTO, Constants, Enums e Utils, que representam, respectivamente, as entidades do sistema, objetos de transferência de dados, constantes de domínio, enumerações e utilitários de manipulação. O Domain é utilizado pelo Core como base para as operações e pelo Infrastructure para persistência e integração.

Por fim, a camada Infrastructure fornece os recursos de suporte técnico e integração externa, contendo os módulos API (para comunicação com serviços externos) e Database (para persistência de dados). Essa camada abstrai detalhes de implementação de acesso a dados, permitindo que o restante do sistema opere de forma independente de tecnologias específicas.

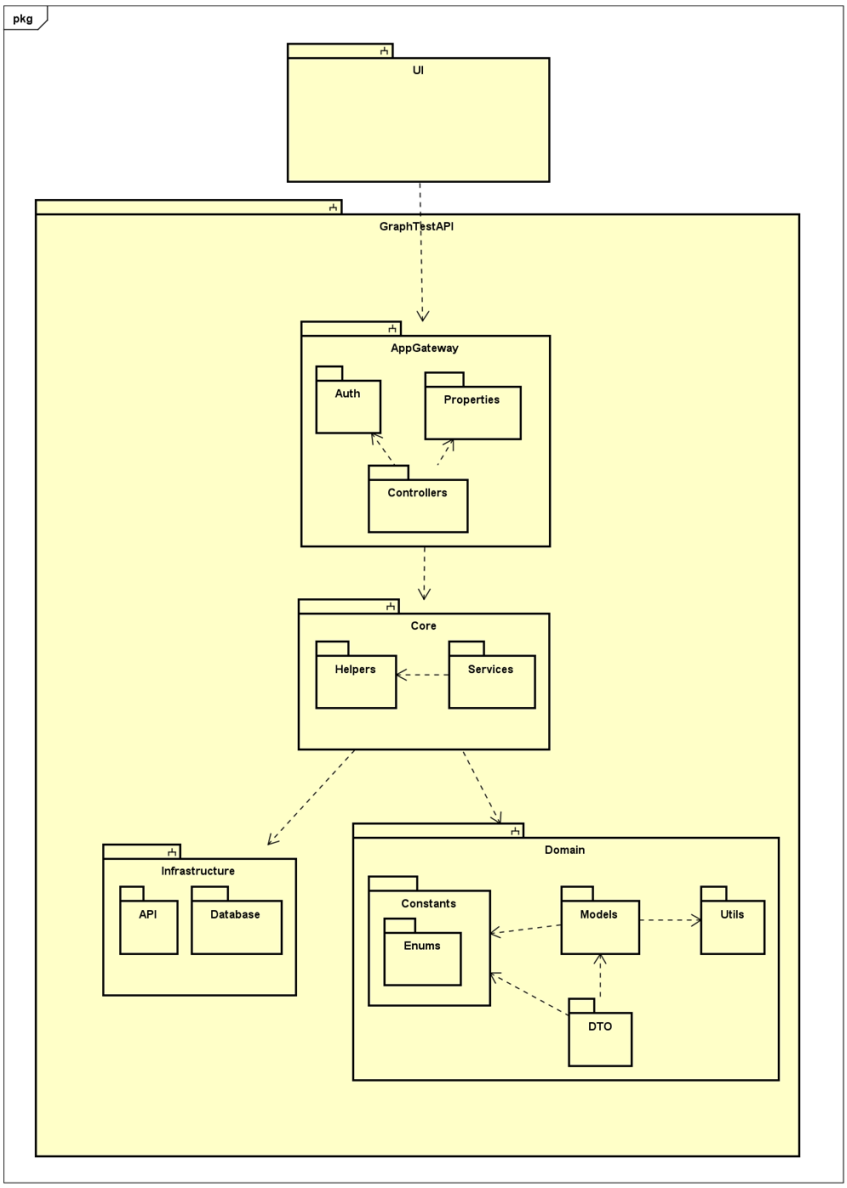


*Figura 15. Diagrama de Arquitetura*

## 3.5 Diagramas de Estados

A Figura 16, Diagrama de Estados do Ciclo de Vida de um GFC (Grafo de Fluxo de Controle) representa o comportamento dinâmico do artefato ao longo de sua criação, análise e cálculo de cobertura. O diagrama descreve as transições de estado desde a disponibilização do código-fonte até a obtenção do relatório final de cobertura, evidenciando as etapas internas do sistema e as ações executadas em cada momento.

O ciclo inicia no estado Código Disponível, em que o sistema registra a fonte do código inserido pelo usuário (entry/registrarFonte()). A partir desse ponto, o usuário pode acionar o comando solicitar gerar GFC, o que leva o sistema ao estado Parsing, no qual o código é analisado

sintaticamente (do/parseToAST()). Caso a análise resulte em erro, a transição ocorre para o estado Exibindo Relatório de Erro, onde são apresentados os diagnósticos e mensagens de falha (entry/exibirDiagnósticos()). O usuário pode então corrigir o código, retornando ao estado Código Corrigido e reiniciando o processo.

Quando o parsing é bem-sucedido ([sucesso]), o sistema passa ao estado Gerando GFC, responsável pela construção efetiva do grafo de fluxo de controle (do/construirCFG()) e sua persistência (exit/persistirGFC()). Em seguida, o artefato entra no estado GFC Gerado, onde é renderizado graficamente na interface (entry/renderizarGrafo()), tornando-se visualizável pelo usuário.

A partir desse ponto, o diagrama apresenta uma bifurcação condicional: caso existam resultados de cobertura disponíveis ([cobertura disponível]), o sistema avança para o estado Calcular Cobertura, no qual são computadas as métricas de cobertura estrutural (do/computarMetricasDeCobertura()) e o relatório correspondente é persistido (exit/persistirRelatorio()). Se não houver dados de cobertura ([cobertura indisponível]), o sistema entra no estado Aguardando Cobertura, exibindo uma mensagem informativa (entry/exibirMensagem("Sem dados de cobertura disponíveis")) e permanecendo em espera até que os resultados sejam importados (exit/importarResultados()).

Quando o processo de cálculo é concluído, o sistema alcança o estado Cobertura Calculada, responsável pela apresentação do relatório final de cobertura (do/exibirRelatorioDeCobertura()), encerrando o ciclo de vida do GFC.

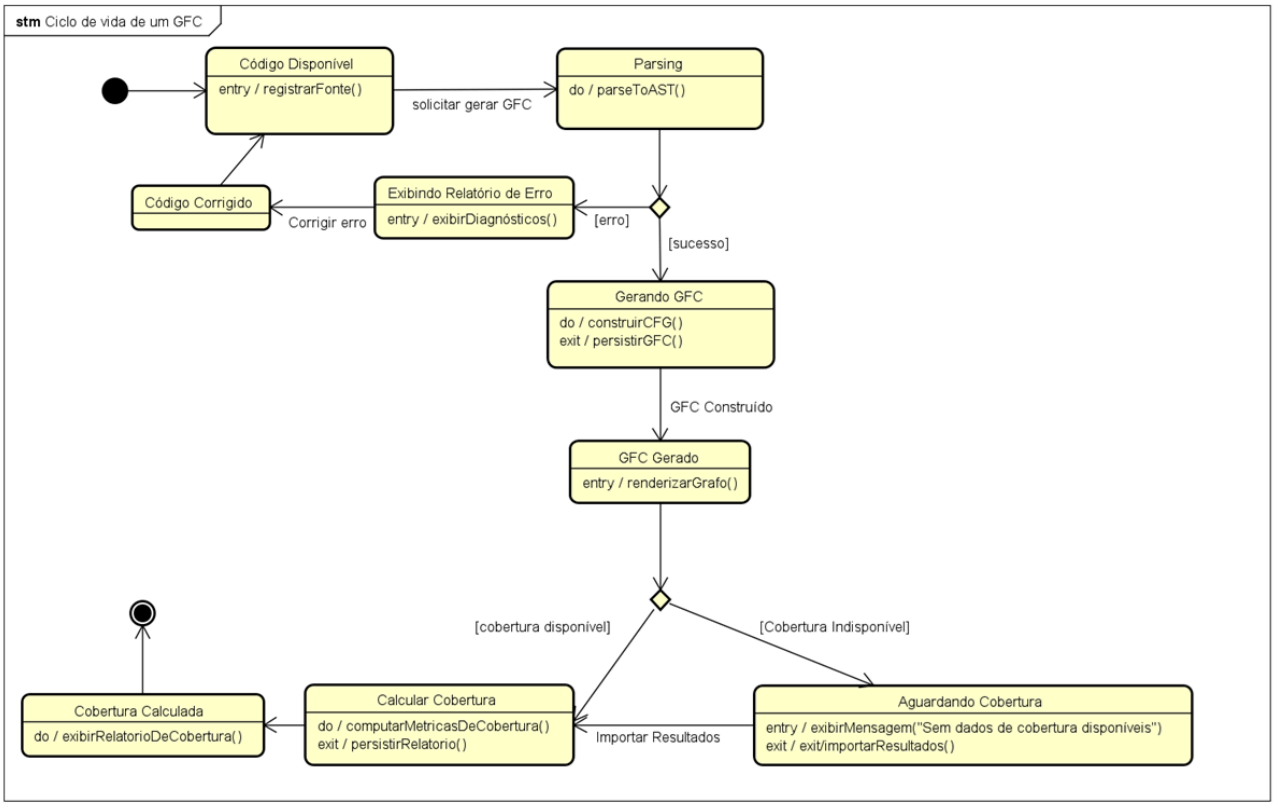

*Figura 16. Diagrama de Estados do Ciclo de Vida do GFC*

A Figura 17, Diagrama de Estados do Ciclo de Vida de um GCE (Grafo de Causa e Efeito) ilustra o comportamento dinâmico do artefato durante seu processo de criação, edição e validação. O diagrama descreve a sequência de estados que o modelo percorre desde sua inicialização até a persistência final, evidenciando as ações internas do sistema e as transições que resultam das interações do usuário ou da verificação automática de consistência lógica.

O ciclo inicia no estado Criado, momento em que o sistema inicializa a estrutura do modelo de GCE (entry/inicializarModelo()) e disponibiliza o editor gráfico ao usuário (do/exibirEditor()). Nesse estado, o modelo encontra-se vazio e apto para receber causas, efeitos e operadores lógicos, como AND, OR e NOT.

Ao adicionar elementos ao grafo, o sistema transita para o estado Em Edição, no qual o modelo passa a ser marcado como alterado (entry/marcarAlterado()) e o resumo das regras é atualizado continuamente conforme o usuário realiza modificações (do/atualizarResumoRegras()). Esse estado reflete o momento ativo de modelagem, em que o GCE é progressivamente construído e ajustado.

Quando o usuário solicita a verificação de consistência do modelo, ocorre a transição para o estado Validando. Nesse ponto, o sistema prepara o modelo para análise lógica (entry/prepararModeloParaSolver()) e executa os procedimentos de verificação (do/verificarConsistencia()), assegurando que todas as relações entre causas e efeitos estejam logicamente corretas e sem contradições.

O resultado dessa validação define a transição seguinte:

- Caso o modelo seja considerado consistente ([sim]), o sistema avança para o estado Salvo, onde o artefato é persistido em banco de dados (entry/persistirModelo()), encerrando o ciclo.

- Caso inconsistências sejam detectadas ([não]), o sistema transita para o estado Exibindo Relatório de Erro, apresentando ao usuário os diagnósticos e mensagens de falha (entry/exibirDiagnósticos()), permitindo a correção manual dos problemas e o retorno ao estado Em Edição.

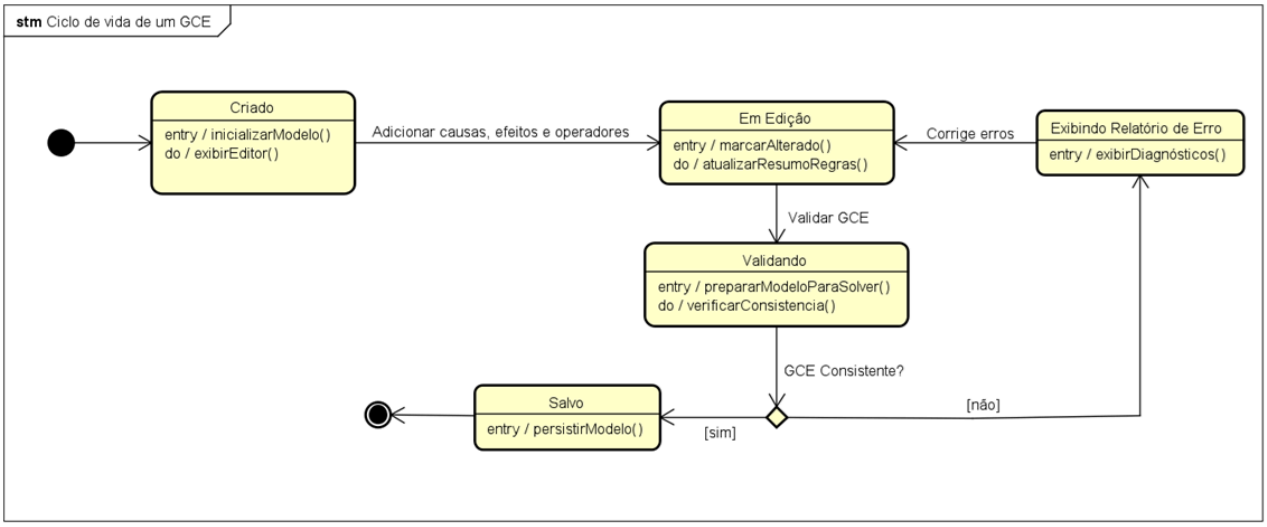


*Figura 17. Diagrama de Estados do Ciclo de Vida do GCE*

## 3.6 Diagrama de Componentes e Implantação.

Nesta seção são apresentados os diagramas de componentes e implantação da plataforma. O diagrama de componentes modela os principais componentes do sistema, apontando suas composições e dependências, bem como as interfaces que eles consomem e que eles proveem. Enquanto isso, o diagrama de implantação representa os recursos físicos e de software necessários para o sistema ser implantado adequadamente.

Na Figura 18 é apresentado o diagrama de componentes do sistema GraphTest. O principal componente representado é o Web Server (Spring Boot API), responsável por prover a API consumida pela aplicação *web*. Esse componente realiza o processamento do código-fonte *Java*, gera o Grafo de Fluxo de Controle e se comunica SGBD por meio do ORM, utilizando SQL para armazenamento de dados.

O Web App (Angular) representa o cliente do sistema, sendo responsável pela interação com o usuário. Ele consome os serviços expostos pelo Web Server e utiliza as bibliotecas CFG Viewer e CEG Editor, ambas baseadas no *Cytoscape.js*, para a visualização e edição dos grafos. Além disso, o módulo Import Module realiza a leitura de arquivos *.java* a partir do Sistema de Arquivos (Local .java / Saídas), enquanto o Export Module é responsável pela exportação das imagens dos grafos gerados.

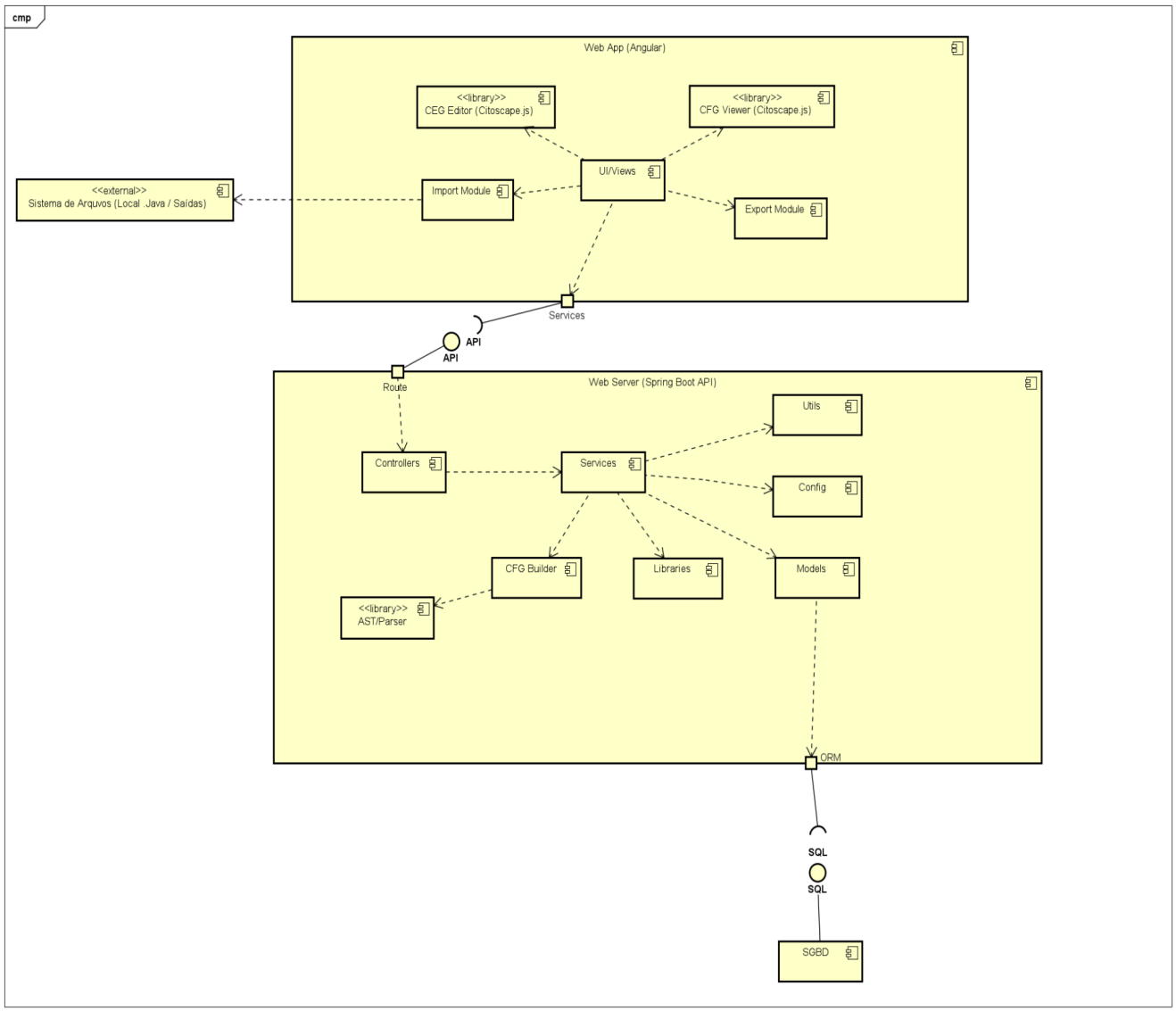


*Figura 18. Diagrama de Componente*

Na Figura 19 é apresentado o diagrama de implantação do sistema GraphTest. Nele são mostrados os nós de processamento utilizados. A aplicação *web* é acessada por meio de um *Web Browser* no *Client*, que obtém os arquivos estáticos do *Web Server – Angular App* via HTTPS e realiza requisições ao *Application Server – VM*, onde está implantada a *Backend Application (Spring Boot API)*. O *Application Server* comunica-se com o *Database Server* e o *File System* por meio do protocolo TCP/IP, responsáveis respectivamente pelo armazenamento de dados e pelos arquivos de entrada e saída da aplicação.

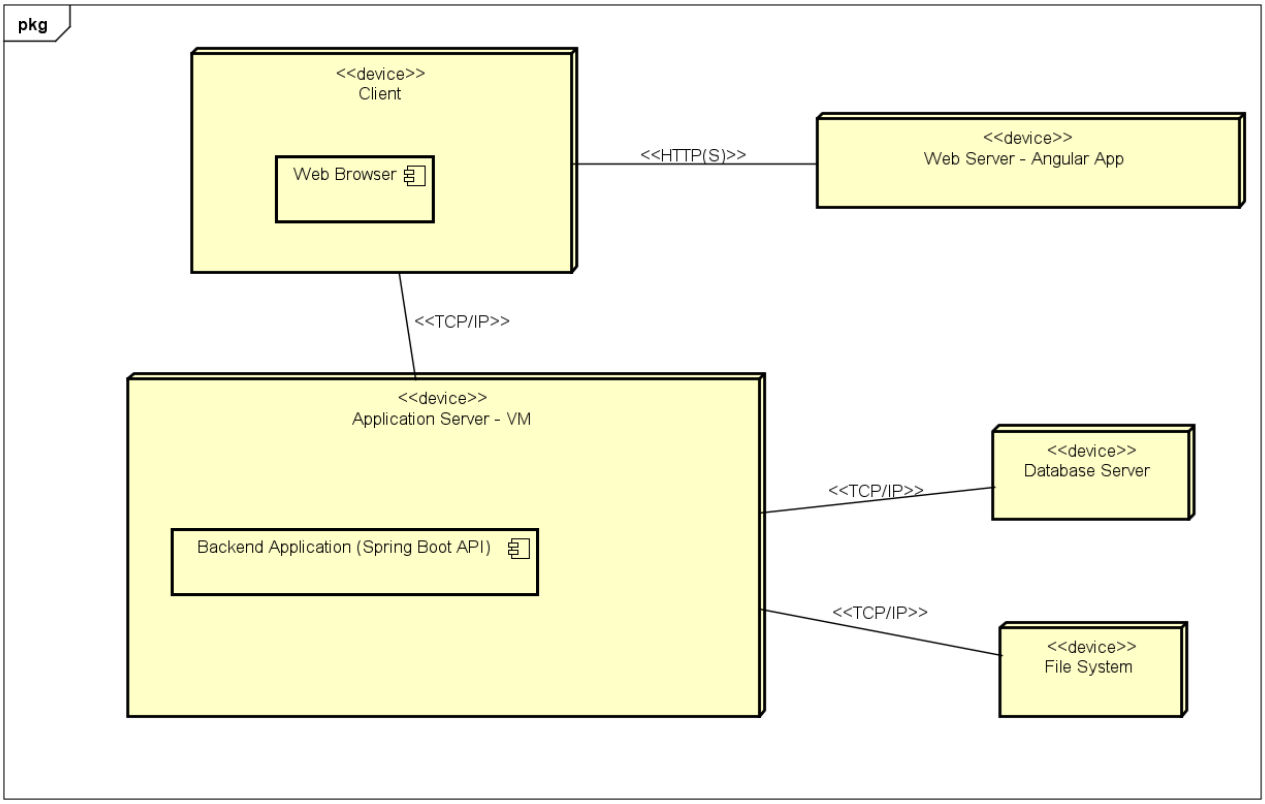


*Figura 19. Diagrama de Implantação*

# 4. Projeto de Interface com Usuário

Nesta seção são apresentados os esboços das interfaces com o usuário. A seção é composta pelo protótipo da Tela Inicial, assim como os protótipos das telas de Projeto, Grafo de Fluxo de Controle, Grafo de Causa e Efeito e da Tabela de Decisão. Todas as interfaces são apresentadas na Seção 4.1, que contém o esboço das telas que serão usadas para todos os atores do sistema.

## 4.1 Esboço das Interfaces Comuns a Todos os Atores

A Figura 20 apresenta o protótipo da página inicial (Home) do sistema GraphTest, concebida para funcionar como o ponto central de navegação e gerenciamento dos artefatos estruturais e funcionais do usuário. A interface foi organizada para oferecer acesso rápido às principais funcionalidades, permitindo que o usuário consulte seus projetos, importe código-fonte, visualize artefatos recentes e inicie fluxos de análise com eficiência.

Na parte superior, a página exibe o cabeçalho de navegação contendo os módulos principais (Projetos, GFC, GCE), seguido de uma seção introdutória com a descrição do propósito da plataforma e dois botões de ação: Criar novo projeto e Importar código. Logo abaixo,

encontra-se o bloco Seus Projetos, que apresenta cartões individuais contendo nome do projeto, data de criação e quantidade de artefatos já associados, além do botão Abrir, permitindo acesso direto ao ambiente interno do projeto.

A seção de Ações Rápidas organiza funcionalidades essenciais em três categorias: geração de GFC, modelagem de GCE e criação de tabelas de decisão. Esses atalhos otimizam o fluxo de trabalho ao permitir que o usuário inicie análises sem navegação adicional. Na sequência, a área Últimos Artefatos Acessados apresenta uma tabela com os artefatos manipulados recentemente, listando nome, tipo (GFC, GCE ou Tabela), data do último acesso e ação correspondente, contribuindo para a retomada rápida de atividades.

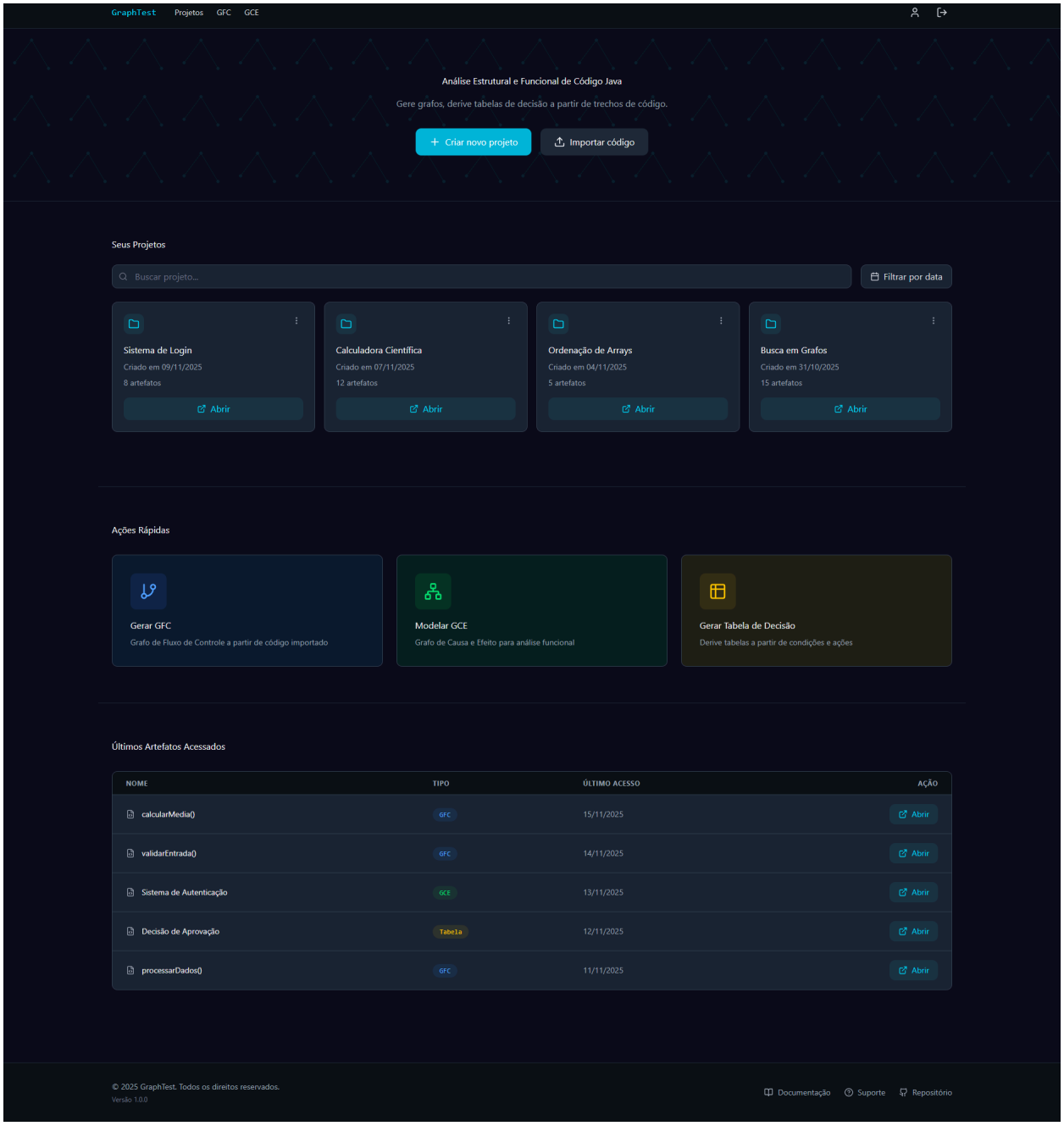


*Figura 20. Protótipo da Tela Inicial da Plataforma*

A Figura 21 apresenta o protótipo da tela de Projeto do sistema GraphTest, responsável por consolidar, em um único ambiente, todas as informações, artefatos e operações relacionadas a um projeto ativo. Essa interface funciona como o painel central de gestão do projeto, permitindo ao usuário acompanhar seu progresso, acessar rapidamente os artefatos gerados e identificar avisos ou inconsistências estruturais e funcionais.

A parte superior da tela exibe o nome do projeto e seu contexto dentro da navegação (“GraphTest > Projetos > Nome do Projeto”), seguida dos botões Editar Projeto e Excluir Projeto, oferecendo controle administrativo sobre o ciclo de vida do projeto. Logo abaixo, é apresentada a barra de navegação interna com as seções: Visão Geral, Artefatos de Código, GFC – Fluxo de Controle, GCE – Causa e Efeito e Tabelas de Decisão, permitindo ao usuário alternar entre os artefatos estruturais.

Na seção Resumo do Projeto, métricas essenciais são exibidas de forma direta: quantidade de artefatos de código importados, GFCs gerados, modelos GCE e tabelas de decisão derivadas. Um indicador de Progresso Geral auxilia o usuário a visualizar o avanço interno do projeto. Informações adicionais, como a data da última atualização e o número de colaboradores, ajudam no acompanhamento do contexto de desenvolvimento.

Ao lado, a área de Ações Rápidas reúne atalhos operacionais para iniciar fluxos principais: gerar um GFC, importar novo arquivo de código-fonte, modelar um GCE e gerar uma TD. Essas ações permitem que o usuário avance rapidamente sem percorrer menus adicionais.

A seguir, a seção Artefatos Recentes exibe cartões com os últimos elementos manipulados no projeto, GFCs, GCEs, tabelas de decisão e arquivos de código, indicando tipo, data e botão de acesso direto. Esse mecanismo facilita a retomada das atividades mais recentes.

Na parte inferior, a área Avisos e Validações apresenta erros, inconsistências e alertas relevantes detectados pelos mecanismos internos do GraphTest, tais como problemas de parsing no GFC, problemas de consistência no GCE e tabelas de decisão incompletas. Cada aviso pode ser expandido para consulta detalhada, apoiando o usuário na correção incremental do projeto.

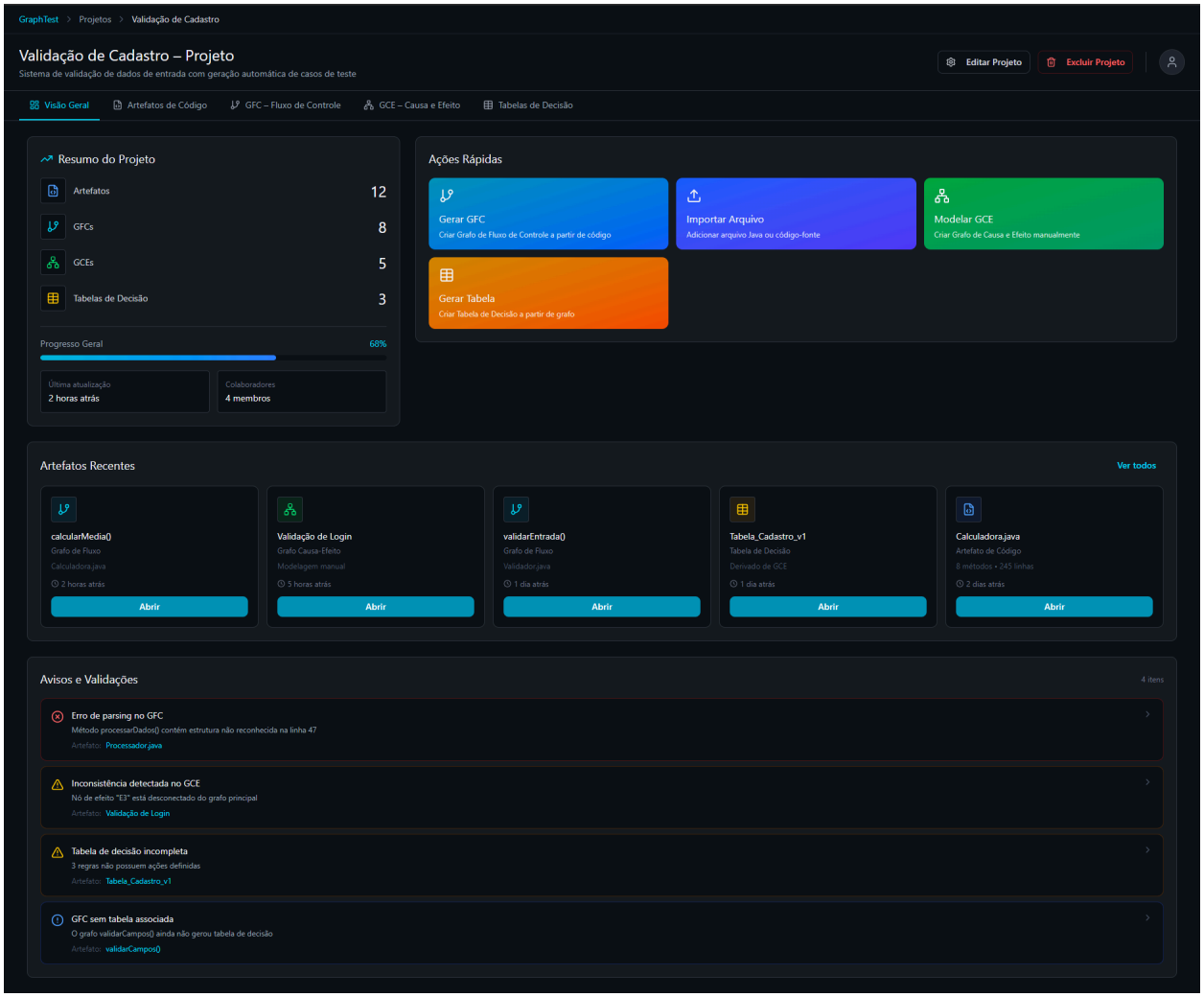


*Figura 21. Protótipo da Tela de um Projeto do usuário*

A Figura 22 apresenta o protótipo da tela de visualização do Grafo de Fluxo de Controle (GFC) no sistema GraphTest, interface dedicada à análise estrutural de métodos Java por meio da representação gráfica de seus nós e arestas. Trata-se de uma das telas centrais do processo de teste estrutural, pois permite ao usuário compreender, inspecionar e navegar pela lógica interna do método analisado.

Na parte superior, a trilha de navegação indica o caminho percorrido pelo usuário (GraphTest > Projeto > GFC > Classe > método), reforçando o contexto do artefato visualizado. A interface disponibiliza também ações rápidas, como Gerar novo GFC e Voltar ao Projeto, mantendo o fluxo de trabalho contínuo.

À esquerda, a coluna Métodos apresenta a lista de métodos da classe selecionada, acompanhados do intervalo de linhas correspondente no arquivo fonte. O usuário pode alternar rapidamente entre diferentes métodos do mesmo artefato, facilitando análises comparativas.

O painel central contém o grafo de fluxo de controle, onde cada nó representa um comando, decisão, laço ou retorno. A tela utiliza uma codificação visual baseada na legenda exibida no canto superior esquerdo (Comando, Decisão, Laço, Retorno), permitindo distinção imediata dos tipos de estruturas. As arestas são marcadas com rótulos lógicos (“true”, “false”), quando aplicável, evidenciando caminhos de execução.

Logo abaixo da legenda, a seção Estatísticas do Grafo apresenta métricas fundamentais: número total de nós, número total de arestas e a complexidade ciclomática derivada automaticamente, indicador essencial para estimar o nível mínimo de testes necessários.

À direita, localiza-se o painel de Informações do Nó, atualizado dinamicamente conforme o nó selecionado no grafo. Esse painel exibe:

- Tipo do nó (ex.: laço, decisão, comando);
- Linha correspondente no código fonte;
- Trecho de código associado;
- Descrição semântica gerada pelo sistema;
- Estado de cobertura (ex.: coberto ou não coberto);
- Botão Destacar no código, permitindo sincronização direta entre visualização gráfica e código-fonte.

Ferramentas adicionais de navegação, zoom in, zoom out e ajuste automático, ficam localizadas na parte inferior direita do grafo, otimizando a interação com estruturas complexas.

No rodapé, são exibidos indicadores de Cobertura de Comandos e Cobertura de Desvios, ambos calculados a partir das trilhas de execução disponibilizadas. Também é mostrado o status da geração do GFC e o timestamp da última atualização, garantindo rastreabilidade.

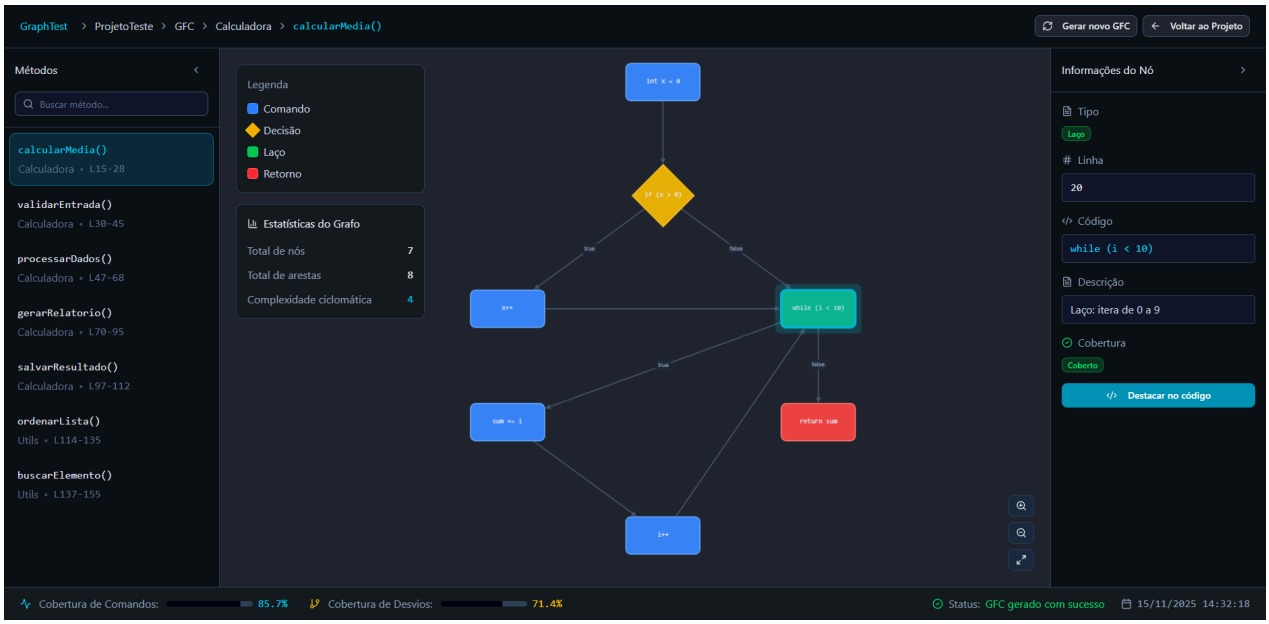


*Figura 22. Protótipo da Tela do Grafo de Fluxo de Controle*

A Figura 23 apresenta o protótipo da tela de edição do Grafo de Causa e Efeito (GCE) no sistema GraphTest, utilizada para a modelagem funcional baseada em causas, efeitos e operadores lógicos. Essa interface fornece um ambiente visual para construir manualmente estruturas de decisão que posteriormente são usadas para derivar tabelas de decisão.

No topo encontra-se a trilha de navegação que orienta o usuário dentro do projeto ativo, seguida dos botões de ação para gerar a TD, validar o grafo e salvar o modelo. Esses comandos representam as operações principais ligadas à finalização e consolidação do GCE.

À esquerda está o painel de ferramentas, onde o usuário dispõe de ações básicas (selecionar, mover e excluir), além de elementos disponíveis para inserção no grafo, como causas, efeitos e operadores lógicos (AND, OR, NOT). Também são apresentadas as opções de criação de arestas, tanto normais quanto negadas, o que permite modelar relações diretas ou relações lógicas invertidas.

A área central constitui o espaço de edição do grafo, estruturado sobre um grid que facilita o posicionamento e alinhamento dos elementos. No exemplo ilustrado, duas causas relacionadas à autenticação do usuário convergem para um operador lógico AND, que por sua vez leva ao efeito representando a permissão de acesso. A liberdade de organização visual permite ao usuário montar desde modelos simples até estruturas mais complexas, mantendo clareza na representação.

À direita, o painel de informações exibe os detalhes do nó selecionado, incluindo seu identificador, tipo, rótulo, tipo de operador (quando aplicável) e coordenadas na área de edição. Esses metadados permitem controle preciso sobre cada elemento, além de possibilitar a persistência e futura restauração do layout.

Ferramentas de zoom e enquadramento, posicionadas na lateral inferior, auxiliam na navegação quando o grafo possui grande extensão. No rodapé, o sistema informa o status das operações realizadas e registra o horário da última modificação.

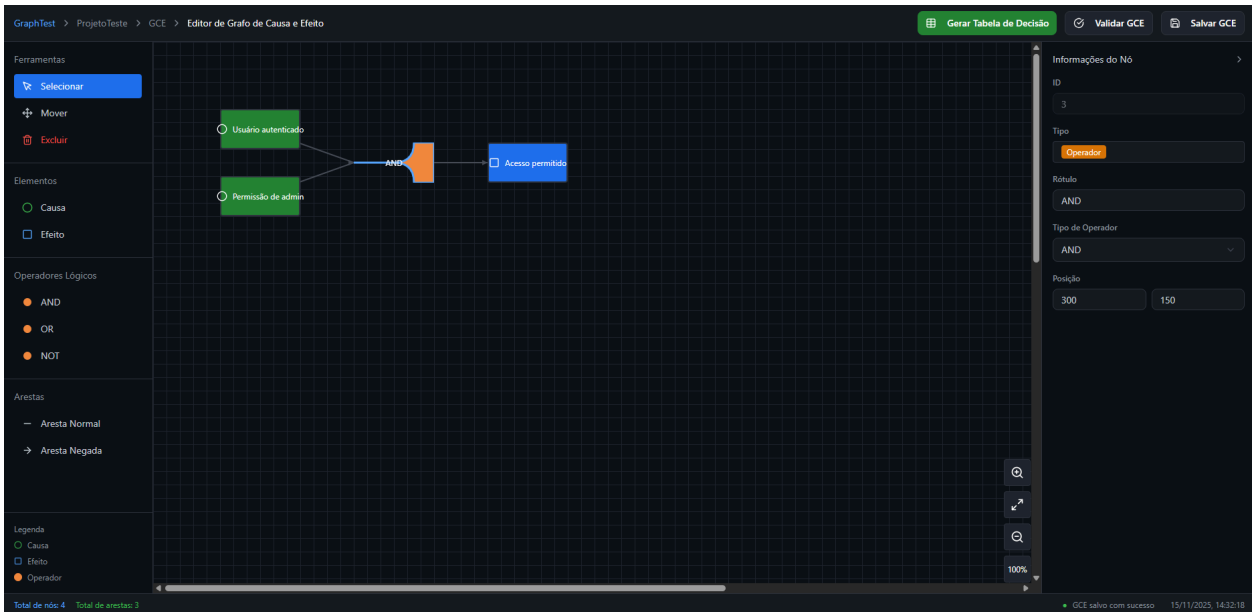


*Figura 23. Protótipo da Tela do Grafo de Causa e Efeito*

A Figura 24 apresenta o protótipo da interface de edição da Tabela de Decisão no GraphTest, utilizada para derivar regras a partir do Grafo de Causa e Efeito associado. A estrutura é organizada para permitir a manipulação simultânea de condições, ações e regras, mantendo alinhamento visual entre todos os elementos da tabela.

No topo da tela está a trilha de navegação que identifica o projeto, o GCE de origem e a tabela atualmente em edição, seguida pelos botões de salvar, reordenar regras e validar a tabela. Essas ações representam as operações centrais do editor e permitem consolidar ou verificar a consistência da estrutura criada.

À esquerda, o painel lateral exibe a estrutura da tabela em duas listas independentes: condições e ações. Cada condição possui um identificador (C1, C2, C3, …) e uma descrição textual; o mesmo ocorre para as ações (A1, A2, A3, …). Esse painel permite adicionar novos elementos ou reorganizar os existentes, definindo a ordem que será refletida no corpo principal da tabela.

A área central contém o grid de regras, onde cada coluna corresponde a uma regra definida (Regra 1, Regra 2, etc.). Para cada condição, o usuário seleciona o valor lógico aplicável àquela regra, “S” (Sim), “N” (Não) ou “–” (irrelevante). A combinação desses valores define o cenário que ativa a regra. As ações aparecem logo abaixo, permitindo o preenchimento do valor associado a cada ação para cada regra, como um booleano, string ou outro tipo permitido.

A interface utiliza cores para reforçar o entendimento: verde para valores verdadeiros, vermelho para valores falsos e cinza para valores neutros, facilitando a leitura rápida das combinações.

Ícones permitem remover regras ou reorganizá-las quando necessário, mantendo flexibilidade na modelagem.

Na parte inferior, o editor apresenta um resumo automático da tabela, indicando quantidade de condições, ações, regras e a política de resolução configurada (por exemplo, “Primeira”). À direita do rodapé, o sistema mostra o status de validação da tabela, indicando se a estrutura gerada está consistente.

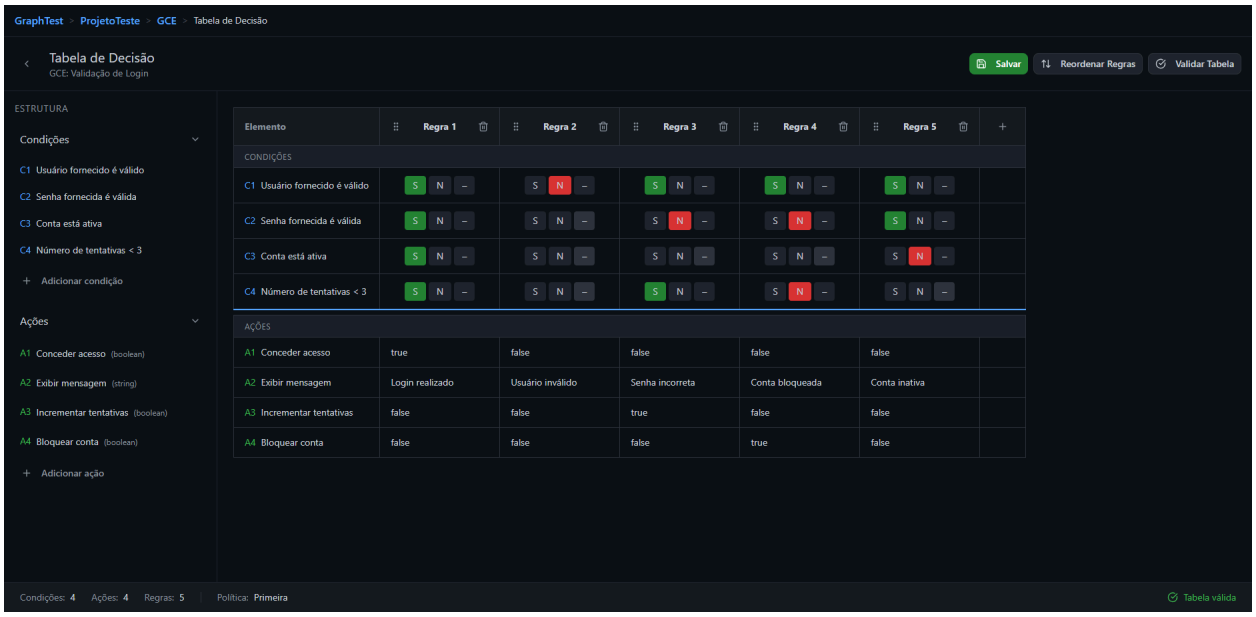


*Figura 24. Protótipo da Tela da Tabela de Decisão*

# 5. Glossário e Modelos de Dados

O glossário apresentado nesta seção descreve os principais atributos reconhecidos pelos usuários do sistema GraphTest, tanto nas entradas de dados (inserção e configuração de grafos, artefatos e testes) quanto nas saídas de dados (relatórios, métricas e visualizações de cobertura). O objetivo é esclarecer o significado de cada termo presente nas interfaces de interação com o usuário.

Os atributos estão organizados de acordo com as principais funcionalidades do sistema, e estão representados nas Tabelas 8 a 14.
A Figura 25 apresenta o modelo de dados (DER) do sistema, evidenciando as entidades, relacionamentos e atributos que dão suporte às informações descritas neste glossário.

| **Autenticação na Plataforma** | | |
|---|---|---|
| **Atributo** | **Formato** | **Descrição** |
| **Nome de Usuário** | Texto (até 120 caracteres) | Nome cadastrado do usuário na plataforma. |
| **E-mail** | Texto (até 160 caracteres) | Endereço de e-mail utilizado para autenticação no sistema. |
| **Senha** | Texto (até 50 caracteres) | Senha inserida pelo usuário para acesso à conta (armazenada de forma criptografada). |

*Tabela 8. Glossário de Autenticação na Plataforma*

| **Gerenciamento de Projetos** | | |
|---|---|---|
| **Atributo** | **Formato** | **Descrição** |
| **Nome do Projeto** | Texto (até 140 caracteres) | Identificação atribuída ao projeto criado pelo usuário. |
| **Descrição do Projeto** | Texto longo | Texto explicativo sobre o propósito e o conteúdo do projeto. |

*Tabela 9. Glossário de Gerenciamento de Projetos*

| **Análise de Código e Geração de Grafo de Fluxo de Controle (GFC)** | | |
|---|---|---|
| **Atributo** | **Formato** | **Descrição** |
| **Código-Fonte** | Texto longo | Código Java importado pelo usuário para análise. |
| **Grafo de Fluxo de Controle (GFC)** | Objeto visual | Representação gráfica gerada automaticamente a partir do código-fonte. |
| **Tipo de Nó** | Lista pré-definida | Classificação do nó (COMANDO, DECISÃO, LAÇO, RETORNO). |
| **Condição da Aresta (Edge Condition)** | Texto curto | Expressão condicional associada à transição entre nós (ex.: “x > 0”). |

*Tabela 10. Glossário de Geração do GFC*

| **Modelagem de Grafo de Causa e Efeito (GCE)** | | |
|---|---|---|
| **Atributo** | **Formato** | **Descrição** |
| **Nome do Grafo GCE** | Texto (até 160 caracteres) | Nome atribuído ao grafo de causa e efeito modelado pelo usuário. |
| **Nome da Causa** | Texto curto | Nome de uma condição lógica (causa) definida no grafo. |
| **Nome do Efeito** | Texto curto | Nome de um resultado esperado (efeito) associado às causas. |
| **Tipo de Operador** | Lista pré-definida | Operador lógico utilizado na modelagem (AND, OR, NOT). |
| **Tipo de Restrição** | Lista pré-definida | Tipo de restrição aplicada às relações (I, E, R, D, L, M). |
| **Status do Grafo** | Lista pré-definida | Situação atual do grafo (EM EDIÇÃO, VALIDADO, INCONSISTENTE). |

*Tabela 11. Glossário do Grafo de Causa e Efeito*

| **Geração da Tabela de Decisão** | | |
|---|---|---|
| **Atributo** | **Formato** | **Descrição** |
| **Tabela de Decisão** | Objeto visual | Tabela derivada automaticamente do GCE, contendo condições e ações. |
| **Índice da Regra (Rule Index)** | Número inteiro | Posição ordinal de uma regra na tabela. |
| **Valor da Condição** | Lista pré-definida | Valor lógico associado à condição (S, N, -). |
| **Valor da Ação** | Lista pré-definida | Valor lógico associado à ação (TRUE, FALSE). |
| **Status da Tabela** | Lista pré-definida | Estado atual da tabela (GERADA, VÁLIDA). |

*Tabela 12. Glossário da Geração da Tabela de Decisão*

| **Assinatura de Teste** | | |
|---|---|---|
| **Atributo** | **Formato** | **Descrição** |
| **Nome do método de teste** | Texto longo | Identificador do método de teste. |
| **Framework de teste** | Texto curto | Ferramenta usada para a criação da assinatura de teste (por exemplo, JUnit, TestNG). |
| **Tipo de teste** | Lista pré-definida | Classificação da assinatura como Teste Estrutural ou Teste Funcional. |
| **Conteúdo do código de teste** | Texto longo | Corpo do método de teste, incluindo preparação, execução e verificações (asserts). |
| **Descrição da assinatura** | Texto curto | Resumo textual explicando o objetivo daquele teste (cenário que ele cobre). |

*Tabela 13. Glossário da Assinatura de Teste*

| **Arquivo Java, Classe, Método e Parâmetros** | | |
|---|---|---|
| **Atributo** | **Formato** | **Descrição** |
| Nome do arquivo Java | Texto curto | Nome do arquivo de código-fonte (por exemplo, `Calculadora.java`). |
| Caminho do arquivo | Texto curto | Localização do arquivo dentro do projeto (por exemplo, `src/main/java/...`). |
| Conteúdo do código | Texto longo | Texto completo do código-fonte Java armazenado e exibido no editor. |
| Nome da classe | Texto curto | Nome da classe principal declarada no arquivo (por exemplo, `Calculadora`). |
| Nome qualificado da classe | Texto curto | Nome completo incluindo pacote (por exemplo, `br.pucmg.graphtest.Calculadora`). |

| | | |
|---|---|---|
| Nome do método | Texto curto | Identificador do método dentro da classe (por exemplo, `calcularMedia`). |
| Modificador do método | Lista pré-definida | Modificador de acesso ou comportamento do método (por exemplo, `public`, `private`, `static`). |
| Tipo de retorno do método | Texto curto | Tipo retornado pelo método (por exemplo, `void`, `int`, `String`). |
| Linha inicial do método | Número inteiro | Número da primeira linha do código em que o método é definido. |
| Linha final do método | Número inteiro | Número da última linha do código em que o método termina. |
| Assinatura do método | Texto curto | Representação compacta do método com nome e tipos de parâmetros (por exemplo, `calcularMedia(int, int)`). |
| Nome do parâmetro | Texto curto | Nome de cada parâmetro declarado na assinatura do método. |
| Tipo do parâmetro | Texto curto | Tipo de dado de cada parâmetro (por exemplo, `int`, `String`, `boolean`). |
| Posição do parâmetro | Número inteiro | Ordem em que o parâmetro aparece na lista de parâmetros do método (primeiro, segundo, terceiro, etc.). |

*Tabela 14. Glossário do Arquivo Java*

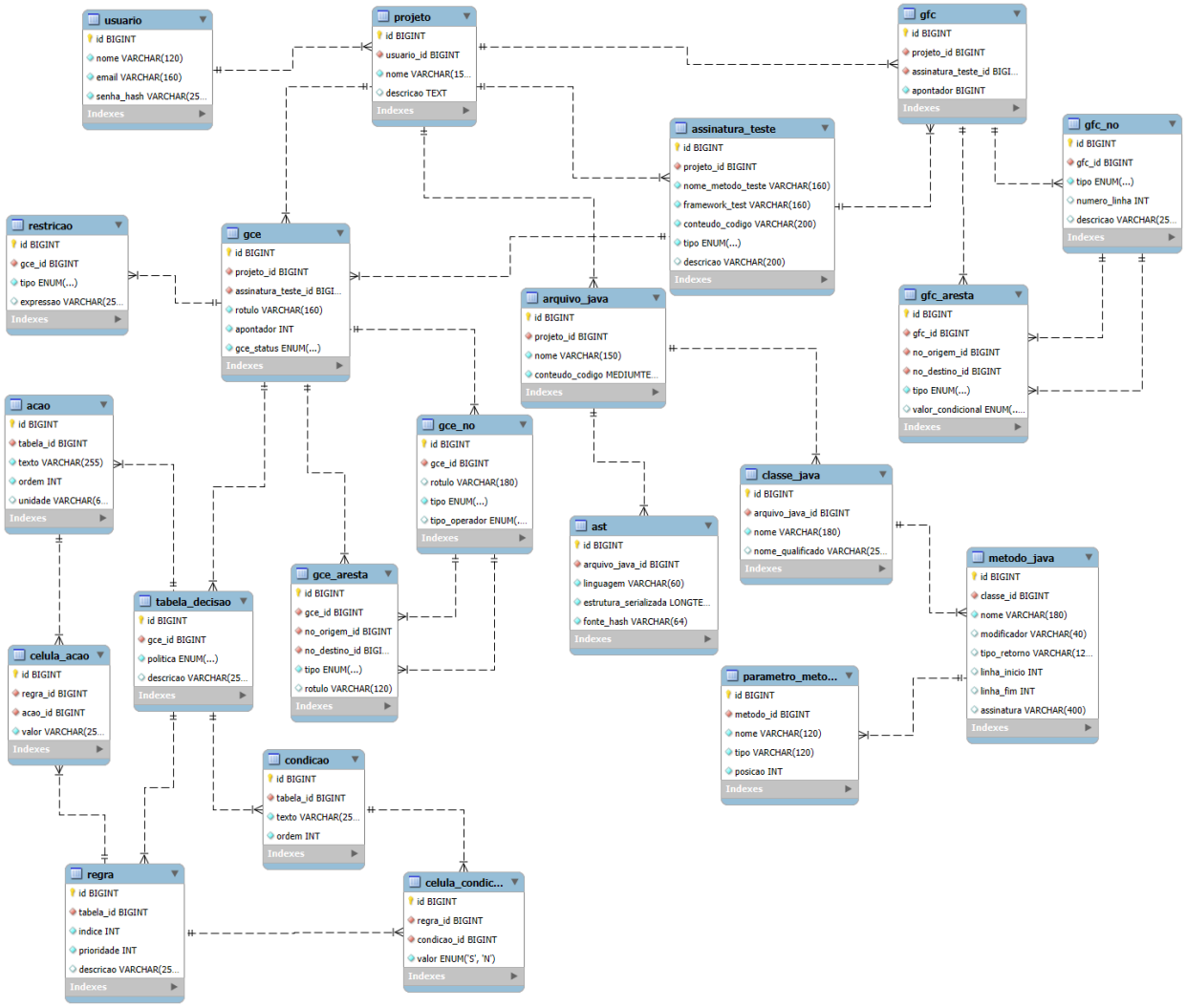


*Figura 25. Diagrama de Entidade e Relacionamento*

# 6. Casos de Teste

Esta seção descreve os casos de teste definidos para a aplicação. A Seção 6.1 apresenta os testes de aceitação derivados das necessidades especificadas no documento de visão, cujo objetivo é verificar se o sistema atende aos requisitos essenciais dos usuários. A Seção 6.2 apresenta o plano de testes de integração, destinado a avaliar o funcionamento conjunto dos componentes do sistema e a garantir que suas interações ocorram de forma correta e consistente.

## 6.1 Teste de Aceitação

Esta seção descreve os testes de aceitação (TA), que visam garantir que o *software* construído atende as necessidades do usuário. Esses testes foram baseados nas necessidades (N) mapeados

no documento de visão. A seguir, são listadas essas necessidades, identificadas pela letra N e uma numeração típica.

**N1.** Gerar GFC automaticamente a partir do código-fonte.
**N2.** Modelar GCE e gerar TD correspondente
**N3.** Visualizar grafos de forma interativa.

Cada caso de teste de aceitação é descrito por meio de uma tabela composta pelas colunas: identificador, necessidade, caso de teste, pré-condições, dados de entrada, ações e resultado esperado. O identificador combina a necessidade e o caso de teste correspondente. A necessidade indica qual requisito o teste valida. O caso de teste nomeia o cenário avaliado. As pré-condições estabelecem o estado necessário antes da execução. Os dados de entrada são as informações fornecidas ao sistema durante o teste. As ações representam os passos executados. O resultado esperado define a resposta que o sistema deve apresentar ao final do teste.

A seguir são apresentados os testes de aceitação relacionados à necessidade N1. A Tabela 15 corresponde ao caso N1T1, que descreve a geração bem-sucedida do GFC a partir de um método válido. A Tabela 16, referente ao caso N1T2, apresenta o comportamento do sistema ao identificar que a classe selecionada não possui métodos. A Tabela 17, correspondente ao caso N1T3, descreve o bloqueio da importação quando o arquivo contém erros de sintaxe. A Tabela 18, por fim, apresenta o caso N1T4, que trata da impossibilidade de utilizar arquivos que não possuem a classe Main.

| | |
|---|---|
| Identificador | N1T1 |
| Necessidade | Gerar GFC automaticamente |
| Caso de teste | Gerar GFC a partir de um método selecionado |
| Pré-condições | • Projeto cadastrado<br>• Arquivo *.java* importado<br>• Arquivo contém ao menos um método compilável |
| Dados de entrada | Arquivo *.java* |
| Ações | • Selecionar a classe<br>• Selecionar o método<br>• Clicar no botão de gerar GFC |
| Dado de resposta | GFC exibido corretamente na interface, contendo nós e arestas correspondentes ao fluxo do método |

*Tabela 15. Teste de aceitação de gerar GFC automaticamente a partir do código-fonte*

| Identificador | N1T2 |
|---|---|
| Necessidade | Gerar GFC automaticamente |
| Caso de teste | Impedir a geração de GFC quando não há métodos disponíveis |
| Pré-condições | • Projeto cadastrado<br>• Arquivo *.java* importado<br>• Arquivo contém ao menos uma classe sem método |
| Dados de entrada | Arquivo *.java* |
| Ações | Selecionar a classe sem método |
| Dado de resposta | Sistema informa que a classe não possui métodos disponíveis e mantém o botão de gerar GFC desabilitado |

*Tabela 16. Teste de aceitação de não gerar GFC de arquivo .java com classe sem método*

| Identificador | N1T3 |
|---|---|
| Necessidade | Gerar GFC automaticamente |
| Caso de teste | Impedir a importação de arquivo .java quando há erro de sintaxe |
| Pré-condições | Projeto cadastrado |
| Dados de entrada | Arquivo *.java* com código inválido para *parsing* |
| Ações | Importar código *.java* |
| Dado de resposta | Sistema retorna mensagem indicando falha no parsing devido a erro de sintaxe |

*Tabela 17. Teste de aceitação de não gerar GFC de arquivo .java com erro de sintaxe*

| | |
|---|---|
| Identificador | N1T4 |
| Necessidade | Gerar GFC automaticamente |
| Caso de teste | Impedir a utilização de arquivo *.java* que não possui a classe *Main* |
| Pré-condições | Projeto cadastrado |
| Dados de entrada | Arquivo *.java* |
| Ações | Importar o arquivo *.java* |
| Dado de resposta | Sistema informa que o arquivo não possui a classe *Main* exigida e não permite prosseguir com o fluxo de geração do GFC |

*Tabela 18. Teste de aceitação de não gerar GFC de arquivo .java sem a classe Main*

A seguir são apresentados os testes de aceitação relacionados à necessidade N2. A Tabela 19 corresponde ao caso N2T1, que descreve o processo de adicionar nós válidos ao GCE durante a modelagem. A Tabela 20 apresenta o caso N2T2, que demonstra o comportamento do sistema ao impedir conexões inválidas entre nós. A Tabela 21 registra o caso N2T3, que trata do bloqueio da criação de restrições inconsistentes. A Tabela 22 detalha o caso N2T4, referente à geração bem-sucedida da tabela de decisão a partir de um GCE completo e válido. Por fim, a Tabela 23 apresenta o caso N2T5, que descreve o bloqueio da validação da tabela de decisão quando há regras incompletas.

| | |
|---|---|
| Identificador | N2T1 |
| Necessidade | Modelar GCE e gerar TD correspondente |
| Caso de teste | Permitir adicionar nós (causas, efeitos e operadores) ao GCE |
| Pré-condições | Projeto cadastrado |
| Dados de entrada | Rótulo, tipo (causa/efeito/operador) e operador lógico quando aplicável |
| Ações | • Iniciar modelagem do GCE<br>• Adicionar nós com seus respectivos |

| | |
|---|---|
| | atributos |
| Dado de resposta | Nós exibidos corretamente no GCE |

*Tabela 19. Teste de aceitação de modelar GCE e gerar TD correspondente adicionando nós válidos ao GCE*

| | |
|---|---|
| Identificador | N2T2 |
| Necessidade | Modelar GCE e gerar TD correspondente |
| Caso de teste | Impedir conexão entre nós quando a validação estrutural for violada |
| Pré-condições | • Projeto cadastrado<br>• GCE iniciado<br>• Pelo menos dois nós adicionados |
| Dados de entrada | Par de nós cuja conexão viola uma regra (ex.: dois efeitos entre si, operador sem operandos, direção inválida) |
| Ações | • Selecionar nó de origem<br>• Selecionar nó de destino<br>• Solicitar conexão entre os nós |
| Dado de resposta | Sistema indica que a conexão é inválida e não cria a aresta no GCE |

*Tabela 20. Teste de aceitação de modelar GCE e gerar TD correspondente impedindo conexões inválidas entre nós*

| | |
|---|---|
| Identificador | N2T3 |
| Necessidade | Modelar GCE e gerar TD correspondente |
| Caso de teste | Impedir a criação de restrição quando os nós envolvidos não estão conectados |
| Pré-condições | • Projeto cadastrado<br>• GCE iniciado<br>• Nós e conexões já modelados |

| | |
|---|---|
| | parcialmente |
| Dados de entrada | IDs de nós selecionados e tipo de restrição (E, I, O, R, M) |
| Ações | • Selecionar os nós para a restrição<br>• Solicitar criação da restrição |
| Dado de resposta | Sistema informa que não é possível criar a restrição porque os nós selecionados não apresentam conexão válida |

*Tabela 21. Teste de aceitação de modelar GCE e gerar TD correspondente impedindo criação de restrições inconsistentes*

| | |
|---|---|
| Identificador | N2T4 |
| Necessidade | Modelar GCE e gerar TD correspondente |
| Caso de teste | Gerar tabela de decisão a partir de um GCE válido |
| Pré-condições | • Projeto cadastrado<br>• GCE modelado e válido |
| Dados de entrada | ID do GCE |
| Ações | Solicitar a geração da tabela de decisão |
| Dado de resposta | TD exibida corretamente na interface, contendo causas, condições, regras, ações e células correspondentes ao GCE |

*Tabela 22. Teste de aceitação de modelar GCE e gerar TD correspondente a partir de um GCE válido*

| | |
|---|---|
| Identificador | N2T5 |
| Necessidade | Modelar GCE e gerar TD correspondente |
| Caso de teste | Impedir a geração da tabela de decisão quando o modelo do GCE apresenta |

| | |
|---|---|
| | inconsistências |
| Pré-condições | • Projeto cadastrado<br>• GCE iniciado e parcialmente modelado<br>• Existência de inconsistências no GCE (ex.: nós desconectados, operador incompleto, restrição mal formada) |
| Dados de entrada | - |
| Ações | Solicitar a geração da tabela de decisão |
| Dado de resposta | Sistema informa que não é possível gerar a tabela de decisão devido a inconsistências no GCE |

*Tabela 23. Teste de aceitação de modelar GCE e gerar TD correspondente impedindo validação da tabela quando há regras incompletas*

A seguir são apresentados os testes de aceitação relacionados à necessidade N3. A Tabela 24 descreve a exibição correta do grafo na interface. A Tabela 25 apresenta o comportamento do sistema ao permitir operações de interação, como zoom, movimentação e seleção de elementos. Por fim, a Tabela 26 registra o bloqueio de interação quando o grafo não possui elementos suficientes para visualização.

| | |
|---|---|
| Identificador | N3T1 |
| Necessidade | Visualizar grafos de forma interativa |
| Caso de teste | Exibir grafo corretamente após sua geração |
| Pré-condições | • Projeto cadastrado<br>• GFC ou GCE previamente gerado e salvo |
| Dados de entrada | - |
| Ações | Abrir o artefato de grafo na interface |
| Dado de resposta | Grafo exibido corretamente na tela, com layout aplicado e elementos visíveis |

*Tabela 24. Teste de aceitação de visualizar grafos de forma interativa exibindo corretamente o grafo na interface*

| | |
|---|---|
| Identificador | N3T2 |
| Necessidade | Visualizar grafos de forma interativa |
| Caso de teste | Permitir operações básicas de interação no grafo exibido |
| Pré-condições | • Projeto cadastrado<br>• Grafo exibido na interface |
| Dados de entrada | - |
| Ações | • Aplicar zoom<br>• Mover o grafo (pan)<br>• Selecionar um nó |
| Dado de resposta | Sistema responde às interações, permitindo ampliar, mover e destacar elementos do grafo |

*Tabela 25. Teste de aceitação de visualizar grafos de forma interativa permitindo operações básicas de interação*

| | |
|---|---|
| Identificador | N3T3 |
| Necessidade | Visualizar grafos de forma interativa |
| Caso de teste | Impedir interação quando o grafo está vazio ou incompleto para exibição |
| Pré-condições | • Projeto cadastrado<br>• Artefato de grafo existente, porém sem elementos visuais (nós/arestas) |
| Dados de entrada | - |
| Ações | Solicitar abertura do grafo na interface |
| Dado de resposta | Sistema informa que o grafo não possui elementos suficientes para visualização e desabilita operações de interação (zoom, pan, seleção) |

*Tabela 26. Teste de aceitação de visualizar grafos de forma interativa impedindo interação quando o grafo está vazio ou incompleto*

## 6.2 Testes de Integração

A seção 6.2 apresenta os casos de teste de integração (TI), cujo objetivo é verificar o comportamento do sistema quando seus componentes internos interagem com módulos externos ou serviços auxiliares. Esses testes validam funcionalidades que dependem de operações de entrada e saída, bem como da troca de informações entre fronteiras do sistema. As tabelas desta seção descrevem, de forma estruturada, todos os elementos necessários para compreender o funcionamento de cada teste de integração.

Cada tabela contém as seguintes colunas: identificador, caso de teste, sistemas envolvidos, interface de comunicação, pré-condições, dados de entrada, ações para iniciar a integração e resultado esperado. O identificador é um número sequencial que distingue cada teste de integração. O caso de teste é uma descrição sucinta do cenário avaliado. Os sistemas envolvidos representam os módulos internos e externos participantes da integração. A interface utilizada especifica o meio pelo qual esses módulos trocam informações. As pré-condições definem o estado inicial necessário antes da execução. Os dados de entrada indicam as informações fornecidas ao sistema. As ações descrevem os passos necessários para iniciar a integração. O resultado esperado define a resposta que o sistema deve apresentar ao final do teste.

A Tabela 27 apresenta o caso de teste de integração destinado a validar o processo de cadastro de novos usuários na plataforma por meio da Interface de Programação de Aplicativo (API, do inglês *Application Programming Interface*). O identificador TI1 corresponde ao caso de teste “Cadastrar usuário na plataforma com dados válidos”, que descreve o cenário no qual o usuário interage com a interface da aplicação preenchendo o formulário de cadastro. O sistema envolvido é a API da plataforma, responsável por receber e processar a requisição enviada pelo *interface*. A interface utilizada para a comunicação é uma requisição Protocolo de Transferência de Estado Representacional (HTTP, do inglês *HiperText Transfer Protocol*) no formato Protocolo de Transferência de Estado Representacional (REST, do inglês *Representational State Transfer*), por meio da qual os dados de cadastro são transmitidos. A pré-condição necessária para a execução do teste consiste na inexistência de outro usuário registrado com o mesmo e-mail. Os dados de entrada incluem o nome, o e-mail e a senha informados pelo usuário, enquanto as ações realizadas abrangem o preenchimento dos campos de cadastro e o acionamento do botão de confirmação. O dado de resposta esperado é a confirmação, retornada pela API, de que o cadastro foi concluído com sucesso.

| | |
|---|---|
| Identificador | TI1 |
| Caso de teste | Cadastrar usuário na plataforma com dados válidos |
| Sistemas envolvidos | API da plataforma |
| Interface | Requisição HTTP no formato REST |

| | |
|---|---|
| Pré-condições | Não existir usuário cadastrado com o mesmo e-mail |
| Dados de entrada | Nome, e-mail e senha |
| Ações | • Preencher o formulário com os dados de cadastro<br>• Clicar no botão de confirmação de cadastro |
| Dado de resposta | Apresentar cadastro realizado com sucesso |

*Tabela 27. Teste de integração de cadastrar usuário na plataforma com dados válidos*

A Tabela 28 apresenta o caso de teste de integração TI2, cujo objetivo é validar o processo de autenticação de um usuário previamente cadastrado na plataforma. O caso de teste "Autenticar usuário com credenciais válidas" representa o cenário em que o usuário acessa a interface do sistema, informa seu e-mail e senha corretos e aciona o botão de login. O sistema envolvido é a API da plataforma, responsável por processar a requisição enviada pela *Interface*. A comunicação ocorre por meio de uma requisição HTTP no formato REST, utilizada para transmitir as credenciais ao backend. A pré-condição para a execução do teste é que o usuário já esteja registrado na base de dados. Os dados de entrada consistem no e-mail e na senha fornecidos pelo usuário. As ações incluem o preenchimento dos campos de login e o acionamento do botão correspondente. O resultado esperado é a autenticação bem-sucedida, evidenciada pela resposta da API contendo o token de acesso, o que confirma que a integração entre a interface e a API ocorreu.

| | |
|---|---|
| Identificador | TI2 |
| Caso de teste | Autenticar usuário com credenciais válidas |
| Sistemas envolvidos | API da plataforma |
| Interface | Requisição HTTP no formato REST |
| Pré-condições | Usuário previamente cadastrado |
| Dados de entrada | e-mail e senha |
| Ações | • Preencher dados de login (e-mail e senha)<br>• Clicar no botão de login |
| Dado de resposta | Apresentar autenticação bem-sucedida acompanhada do token de acesso |

*Tabela 28. Teste de integração de login do usuário com credenciais válidas*

A Tabela 29 apresenta o caso de teste de integração TI3, cujo objetivo é validar o comportamento da plataforma quando o usuário tenta realizar login utilizando credenciais inválidas. O caso de teste “Impedir autenticação com credenciais inválidas” representa a situação em que o usuário acessa a tela de login, informa um e-mail inexistente ou uma senha incorreta e aciona o botão de autenticação. O sistema envolvido é a API da plataforma, responsável por receber e processar a requisição enviada pela interface por meio de uma requisição HTTP no formato REST. A pré-condição para a execução do teste é a existência de um usuário cadastrado na base, garantindo que o erro se origina das credenciais informadas e não da ausência de registro. Os dados de entrada consistem em um conjunto de credenciais incorretas informadas pelo usuário. A ação realizada é o acionamento do botão de login, que envia a requisição para a API. O dado de resposta esperado é a indicação, por parte da API, de que a autenticação falhou devido às credenciais inválidas, demonstrando que o mecanismo de validação está funcionando corretamente e que o sistema trata adequadamente tentativas de acesso indevidas.

| | |
|---|---|
| Identificador | TI3 |
| Caso de teste | Impedir autenticação com credenciais inválidas |
| Sistemas envolvidos | API da plataforma |
| Interface | Requisição HTTP no formato REST |
| Pré-condições | Usuário existente na base da plataforma |
| Dados de entrada | E-mail ou senha incorretos preenchidos pelo usuário |
| Ações | Clicar no botão de login |
| Dado de resposta | Apresentar que a autenticação falhou por credenciais inválidas |

*Tabela 29. Teste de integração de login do usuário com credenciais inválidas*

A Tabela 30 apresenta o caso de teste de integração TI4, cujo objetivo é validar o processo completo de geração do GFC a partir de um método selecionado pelo usuário, bem como a obtenção automática da complexidade ciclomática associada a esse grafo. O caso de teste "Gerar GFC a partir de um método selecionado" descreve o cenário em que o usuário, após autenticar-se na plataforma e importar um arquivo .java, seleciona uma classe é um método disponibilizado na interface para iniciar a geração do grafo. O sistema envolvido é a API da plataforma, responsável por receber a requisição enviada via HTTP no formato REST e acionar os módulos internos responsáveis pelo *parsing* do código fonte, derivação do GFC e cálculo da complexidade ciclomática. As pré-condições incluem o usuário autenticado, o projeto criado, o arquivo .java já importado no projeto e a disponibilidade da classe e do método para seleção. Os dados de entrada consistem na escolha da classe e do método exibidos na interface. As ações realizadas pelo usuário incluem selecionar a classe, selecionar o método desejado e clicar no botão "Gerar GFC". O resultado esperado é que a API retorne o grafo correspondente ao método selecionado e apresente, junto ao GFC exibido na interface, o valor da complexidade ciclomática calculada automaticamente, permitindo ao usuário visualizar tanto a estrutura do fluxo de controle quanto sua métrica estrutural associada.

| | |
|---|---|
| Identificador | TI4 |
| Caso de teste | Gerar GFC a partir de um método selecionado |
| Sistemas envolvidos | API da plataforma |
| Interface | Requisição HTTP no formato REST |
| Pré-condições | ● Usuário autenticado na plataforma<br>● Projeto cadastrado<br>● Arquivo *.java* importado<br>● Classe e método disponíveis para seleção |
| Dados de entrada | Seleção de classe e método exibidos na interface |
| Ações | ● Selecionar a classe desejada<br>● Selecionar o método desejado<br>● Clicar no botão "Gerar GFC" |
| Dado de resposta | Apresenta GFC correspondente ao método selecionado e sua complexidade ciclomática calculada |

*Tabela 30. Teste de integração de gerar GFC a partir de um método válido*

A Tabela 31 apresenta o caso de teste de integração TI5, cujo objetivo é validar o processo de geração do esqueleto de teste estrutural a partir de um GFC previamente construído. O caso de teste “Gerar esqueleto de teste estrutural a partir do GFC” descreve o cenário no qual o usuário, após autenticar-se na plataforma, importar um arquivo .java e gerar o GFC para um método específico, aciona a funcionalidade responsável por derivar a estrutura inicial do teste. O sistema envolvido é a API da plataforma, que recebe a requisição enviada pela interface por meio de uma requisição HTTP no formato REST. As pré-condições para esse teste incluem a existência de um usuário autenticado, projeto criado, um arquivo .java já importado no projeto e o respectivo GFC já gerado com sucesso. Como não há necessidade de fornecer novos dados de entrada, o teste se concentra nas ações do usuário, que consistem em acionar o botão de geração do esqueleto de teste. O resultado esperado é que a API retorna o esqueleto de teste estrutural derivado do GFC e que esse conteúdo seja exibido na interface, confirmando que a integração entre o módulo responsável pelo grafo e o módulo de derivação de testes está funcionando corretamente.

| | |
|---|---|
| Identificador | TI5 |
| Caso de teste | Gerar esqueleto de teste estrutural a partir do GFC |
| Sistemas envolvidos | API da plataforma |
| Interface | Requisição HTTP no formato REST |
| Pré-condições | ● Usuário autenticado na plataforma<br>● Projeto cadastrado<br>● Arquivo *.java* importado<br>● GFC gerado |
| Dados de entrada | - |
| Ações | Clicar no botão de gerar esqueleto de teste |
| Dado de resposta | Apresenta o esqueleto de teste derivado do GFC |

*Tabela 31. Teste de integração de gerar esqueleto de teste estrutural*

A Tabela 32 apresenta o caso de teste de integração TI6, cujo objetivo é validar o processo de modelagem do GCE por meio da interface da plataforma, incluindo a verificação final de consistência do modelo construído. O caso de teste “Modelar GCE” descreve a interação do usuário com a interface de modelagem, na qual ele adiciona nós do tipo causa, efeito ou operador, bem como conexões e regras necessárias para estruturar o grafo. O sistema envolvido é a API da plataforma, que recebe as requisições enviadas pela interface por meio de comunicação baseada em requisições HTTP no formato REST. As pré-condições para a execução do teste incluem o usuário autenticado, a existência de um projeto cadastrado e a inicialização do GCE na

interface. Os dados de entrada compreendem os rótulos, tipos e seleções fornecidos pelo usuário ao adicionar nós e conexões. As ações incluem clicar no botão de adicionar nós e clicar no botão de adicionar regras, o que resulta em chamadas REST enviadas à API. O dado de resposta esperado é que a plataforma retorne o GCE estruturado com sua consistência validada, assegurando que não existam erros de conectividade ou lógica no grafo modelado.

| | |
|---|---|
| Identificador | TI6 |
| Caso de teste | Modelar GCE |
| Sistemas envolvidos | API da plataforma |
| Interface | Requisição HTTP no formato REST |
| Pré-condições | ● Usuário autenticado na plataforma<br>● Projeto cadastrado<br>● GCE inicializado |
| Dados de entrada | Rótulos, tipos e seleções feitas pelo usuário ao adicionar nós e conexões |
| Ações | ● Clicar no botão de adicionar nós<br>● Clicar no botão para adicionar regras |
| Dado de resposta | Apresenta GCE com consistência validada |

*Tabela 32. Teste de integração de modelar GCE*

A Tabela 33 apresenta o caso de teste de integração TI7, cujo objetivo é validar o processo de geração da tabela de decisão a partir de um GCE previamente modelado e validado pelo usuário. O caso de teste “Gerar tabela de decisão” descreve o cenário em que o usuário, por meio da interface da plataforma, aciona a funcionalidade responsável por derivar automaticamente a tabela de decisão com base na estrutura lógica do GCE. O sistema envolvido é a API da plataforma, que recebe a solicitação por meio de uma requisição HTTP no formato REST. As pré-condições para execução incluem o usuário autenticado, um projeto cadastrado, o GCE completamente modelado e sua consistência já validada. Os dados de entrada consistem na identificação do GCE selecionado pelo usuário na interface. As ações descritas envolvem acessar a tela de modelagem do GCE e clicar no botão “Gerar Tabela de Decisão”. O resultado esperado é que a API retorna a tabela de decisão devidamente derivada, contendo as condições, regras e ações correspondentes, exibindo-a na interface de forma estruturada e consistente com o modelo fornecido.

| | |
|---|---|
| Identificador | TI7 |
| Caso de teste | Gerar tabela de decisão |
| Sistemas envolvidos | API da plataforma |
| Interface | Requisição HTTP no formato REST |
| Pré-condições | ● Usuário autenticado na plataforma<br>● Projeto cadastrado<br>● GCE modelado |
| Dados de entrada | Identificação do GCE selecionado pelo usuário |
| Ações | Clicar no botão para gerar tabela de decisão |
| Dado de resposta | Apresenta a tabela de decisão derivada do GCE e a exibe na interface com condições, regras e ações geradas |

*Tabela 33. Teste de integração de gerar tabela de decisão*

A Tabela 34 apresenta o caso de teste de integração TI8, cujo objetivo é validar o processo de geração do esqueleto de teste funcional a partir de um GCE previamente modelado e validado pelo usuário. O caso de teste “Gerar esqueleto de teste funcional a partir de um GCE validado” descreve o cenário em que o usuário, por meio da interface da plataforma, aciona a funcionalidade responsável por derivar automaticamente a estrutura inicial de um teste funcional com base nos efeitos identificados no GCE. O sistema envolvido é a API da plataforma, que recebe a solicitação por meio de uma requisição HTTP no formato REST. As pré-condições necessárias incluem o usuário autenticado, a existência de um projeto cadastrado, o GCE totalmente modelado e a confirmação de que sua consistência foi validada. Os dados de entrada são representados pela seleção do GCE realizada pelo usuário. As ações consistem em acessar a interface do GCE e clicar no botão “Gerar Teste Funcional”. O resultado esperado é que a API retorna o esqueleto de teste funcional derivado do GCE validado, exibindo o conteúdo na interface.

| | |
|---|---|
| Identificador | TI8 |
| Caso de teste | Gerar esqueleto de teste funcional |
| Sistemas envolvidos | API da plataforma |
| Interface | Requisição HTTP no formato REST |
| Pré-condições | ● Usuário autenticado na plataforma |

| | |
|---|---|
| | • Projeto cadastrado<br>• GCE modelado |
| Dados de entrada | Identificação do GCE selecionado pelo usuário |
| Ações | Clicar no botão para gerar esqueleto de teste |
| Dado de resposta | Apresenta o esqueleto de teste funcional derivado do GCE validado |

*Tabela 34. Teste de integração de gerar esqueleto de teste funcional*

# 7. Cronograma e Processo de Implementação

A Seção 7 apresenta a organização temporal e operacional do desenvolvimento do sistema. A Seção 7.1 descreve o cronograma de implementação, estruturado em *sprints* quinzenais que distribuem as tarefas previstas ao longo dos quatro meses e meio de desenvolvimento. A Seção 7.2 detalha o processo de implementação, incluindo o fluxo iterativo de desenvolvimento, o uso de *issues* e do quadro *Kanban* no *GitHub* para organização das tarefas, e o emprego de integração contínua para garantir que cada incremento aprovado seja automaticamente integrado e disponibilizado no ambiente de desenvolvimento.

## 7.1 Cronograma

O período de desenvolvimento do projeto é de cerca de quatro meses e meio, iniciando em 1º de fevereiro de 2026 e encerrando em 16 de junho de 2026. O cronograma é organizado em marcos quinzenais (*Milestones*), estruturados para permitir acompanhamento contínuo e revisões frequentes. A Tabela 27 detalha cada *Sprint*, seus respectivos intervalos e as atividades previstas para cada etapa.

| Nome | Período | Atividades | |
|---|---|---|---|
| | | Maria Eduarda Amaral | Lucas Cabral |
| *Sprint 01* | 01/02/2026 - 15/02/2026 | Implementar o fluxo completo de autenticação (UC01) no front-end; construir telas de cadastro e login; construir tela de projeto (UC02); validar interações e regras de | Implementar o backend do UC01 e UC02; criar endpoints de cadastro, login e manutenção de projetos; configurar segurança, sessões |

| | | | |
|---|---|---|---|
| | | acesso; integrar as chamadas à API. | e validação de tokens; estruturar o CRUD de projetos. |
| *Sprint 02* | 16/02/2026 – 02/03/2026 | Desenvolver telas para importação de código e visualização inicial da árvore de arquivos; mapear a interação da interface do UC03 e UC04. | Implementar importação de arquivos Java; gerar e armazenar AST; criar serviços que alimentam a visualização da estrutura do código; integrar AST e cadastro de artefatos ao projeto. |
| *Sprint 03* | 03/03/2025 - 17/03/2025 | Construir interface para apresentação do grafo de fluxo de controle (GFC); implementar visualização dos nós e arestas; incorporar legenda e estatísticas do grafo. | Implementar geração do GFC a partir da AST (UC03); construir lógica da complexidade ciclomática (UC06); gerar assinatura do teste estrutural (UC05); disponibilizar dados para o front. |
| *Sprint 04* | 18/03/2025 - 01/04/2025 | Implementar o editor visual do GCE (UC07): criação de nós, operadores, efeitos e ligações; controles de seleção, exclusão e movimentação. | Implementar backend do GCE: cadastro de nós, arestas e operadores; validar consistência estrutural (UC09); criar endpoints de validação e salvamento. |
| *Sprint 05* | 02/04/2025 - 16/04/2025 | Desenvolver tela completa de tabela de decisão: condições, ações, matriz de regras e edição dos valores. | Implementar conversão automática do GCE em tabela de decisão (UC10); |

| | | | |
|---|---|---|---|
| | | | gerar regras, condições e ações; aplicar políticas de conflito; validar a estrutura final. |
| *Sprint 06* | 17/04/2025 - 01/05/2025 | Aprimorar telas do GFC, GCE e tabela de decisão com base nos fluxos finais; ajustar navegação e usabilidade; refinar elementos visuais e interações. | Realizar melhorias no backend com base no uso integrado; revisar cálculos, estruturas e modelos; corrigir falhas derivadas da integração dos três módulos. |
| *Sprint 07* | 02/05/2025 - 16/05/2025 | Trabalhar nos ajustes solicitados após testes internos; reavaliar responsividade das interfaces; corrigir inconsistências de interação. | Aplicar correções derivadas de feedback da primeira versão estável; otimizar consultas; melhorar performance geral; reorganizar algumas partes do código. |
| *Sprint 08* | 17/05/2025 - 31/05/2025 | Refinar fluxos; validar visuais e interações finais antes da implantação. | Pagar dívidas técnicas acumuladas; revisar módulos sensíveis; estabilizar integrações; preparar build definitivo para produção. |
| *Sprint 09* | 01/06/2025 - 16/06/2025 | Realizar testes finais pós-implantação; ajustar pontos de UX observados em produção. | Implantar o sistema; monitorar funcionamento; corrigir falhas emergenciais; redigir o post-mortem. |

*Tabela 35. Cronograma*

## 7.2 Processo de Implementação

O processo de implementação adotado segue um fluxo incremental e iterativo, estruturado para garantir alinhamento contínuo entre requisitos, modelos de projeto e componentes entregues. A implementação é conduzida em ciclos curtos, cada um responsável por entregar partes funcionais completas, priorizando a rastreabilidade com os casos de uso definidos e a validação progressiva de cada módulo.

A execução inicia pela infraestrutura mínima do sistema, como autenticação, controle de acesso e gerenciamento de projetos, pois esses elementos constituem a base operacional sobre a qual os demais módulos dependem. Após estabilizar o núcleo de acesso (UC01 e UC02), o processo avança para os módulos analíticos estruturais, implementando a importação de arquivos *Java*, a geração da AST e a construção do GFC (UC03, UC04, UC05 e UC06). Em seguida, o foco se desloca para o núcleo funcional, com a implementação do editor de GCE (UC07, UC09) e do mecanismo de derivação da TD (UC10). Em todos os casos, a ordem de trabalho se mantém: definição do modelo de domínio, implementação dos serviços e repositórios, exposição de endpoints e, por fim, integração com o front-end conforme os protótipos de interface definidos.

A organização das tarefas de desenvolvimento é realizada por meio de *issues* no *GitHub*, estruturadas em um quadro *Kanban*. Cada funcionalidade, correção ou melhoria é registrada como *issue*, associada a um caso de uso ou requisito específico, e movimentada entre colunas que representam os estados do fluxo de trabalho (por exemplo, "*Backlog*", "Em andamento", "Em revisão" e "Concluído"). Esse quadro funciona como mecanismo central de controle de progresso, permitindo visualizar o andamento das *sprints*, identificar gargalos e manter a rastreabilidade entre atividades de implementação, modelos de projeto e requisitos elicitados.

O processo de desenvolvimento é suportado por Integração Contínua configurada na própria plataforma *GitHub*. Cada funcionalidade implementada é desenvolvida em *branch* específica, vinculada às *issues* em aberto. Ao término da implementação, o código é submetido via *pull request*, acionando uma *pipeline* automatizada. Essa *pipeline* executa etapas de *build* do *back-end* e do *front-end*, valida dependências, roda testes automatizados disponíveis e aplica verificações básicas de qualidade de código. Em caso de sucesso, o *merge* é autorizado e o artefato resultante é implantado automaticamente no ambiente de desenvolvimento, garantindo que a cada incremento aprovado exista uma versão executável e atualizada do sistema para testes integrados.

Os ciclos de integração entre *front-end* e *back-end* são planejados para ocorrer de forma contínua ao longo das *sprints*. A cada conjunto de funcionalidades concluídas e integrado ao ambiente de desenvolvimento, são realizados testes exploratórios e de aceitação, com foco na coerência entre os artefatos visuais (GFC, GCE e TD) e os comportamentos definidos nos diagramas de sequência, comunicação e estado. Esses testes alimentam novas *issues* de correção e refinamento, realimentando o quadro *Kanban* e fechando o ciclo iterativo.

Na etapa final, o processo concentra-se na estabilização da aplicação, eliminação de dívidas técnicas e ajuste fino de desempenho e usabilidade. As últimas *sprints* são direcionadas à correção de falhas identificadas em ambiente de desenvolvimento, ao refinamento das interfaces e à consolidação de *logs* e mecanismos de monitoramento básico. Concluídas essas atividades, o sistema é preparado para implantação estável, acompanhada por testes finais e por um registro pós-mortem que sintetiza decisões, problemas encontrados e melhorias planejadas para evoluções futuras.

# Referências

UNSPLASH. Persona. Disponível em: <https://unsplash.com/pt-br/s/fotografias/persona>. Acesso em: 18 nov. 2025.