
Documentação de Projeto

para o sistema

GitGrade

Versão 7.0

Projeto de sistema elaborado pelo(s) aluno(s) Guilherme Gabriel Silva Pereira e Lucas Ângelo Oliveira Martins Rocha e apresentado ao curso de **Engenharia de Software** da **PUC Minas** como parte do Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo do professor Cleiton Silva Tavares, orientação acadêmica do professor José Laerte Pires Xavier Junior e orientação de TCC II do professor Cleiton Silva Tavares.

26 de novembro de 2023.

Tabela de Conteúdo

Tabela de Conteúdo	2
Histórico de Revisões	3
1. Introdução	6
2. Modelos de Usuário e Requisitos	7
2.1 Descrição de Atores	8
2.2 Modelos de Usuários	8
2.3 Modelo de Casos de Uso e Histórias de Usuários	10
2.3.1 Diagrama de Casos de Uso	10
2.3.2 Histórias de Usuários	12
2.4 Diagrama de Sequência do Sistema e Contrato de Operações	14
3. Modelos de Projeto	24
3.1 Diagrama de Classes	25
3.2 Diagramas de Sequência	27
3.3 Diagramas de Comunicação	33
3.4 Arquitetura	39
3.5 Diagramas de Estados	42
3.6 Diagrama de Componentes e Implantação	43
4. Projeto de Interface com Usuário	45
5. Glossário e Modelos de Dados	59
6. Casos de Teste	64
6.1 Teste de Aceitação	65
6.2 Testes de Integração	79
7. Riscos	86
8. Cronograma e Processo de Implementação	87
8.1 Cronograma	87
8.2 Processo de Implementação	95
9. Post-mortem	97
9.1 Experiência Positivas	98
9.2 Experiência Negativas	99
9.3 Lições Aprendidas	100
9.4 Repositório do Trabalho	101

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Guilherme Gabriel e Lucas Ângelo	05/03/2023	<ul style="list-style-type: none"> • Criação do documento; • Inserindo título, nomes e datas; • Adicionando a Seção 1. Introdução; • Adicionando a Seção 2.3. Modelo de Casos de Uso e Histórias de Usuários; • Adicionando a Seção 2.3.1. Diagrama de Casos de Uso; • Adicionando a Seção 2.3.2. Histórias de Usuários. 	1.0
Lucas Ângelo	10/03/2023	<ul style="list-style-type: none"> • Corrigindo Diagrama de Casos de Uso, com adição dos atores secundários GitHub e Aluno; • Corrigindo erros gramaticais e removendo verbos no futuro. 	1.1
Lucas Ângelo	13/03/2023	<ul style="list-style-type: none"> • Adicionando a Seção 4. Projeto de Interface com Usuário. 	1.2
Guilherme Gabriel	14/03/2023	<ul style="list-style-type: none"> • Adicionando a Seção 2. Modelos de Usuário e Requisitos; • Adicionando a Seção 2.1. Descrição de Atores; • Adicionando a Seção 2.2. Modelos de Usuários. 	1.3
Lucas Ângelo	07/04/2023	<ul style="list-style-type: none"> • Corrigindo problemas apontados no diagrama de caso de uso, interfaces de usuário e erros gramaticais; • Adicionando casos de uso faltantes; • Adicionando a Seção 2.4. Diagrama de Sequência do Sistema e Contrato de Operações. 	2.0
Lucas Ângelo	11/04/2023	<ul style="list-style-type: none"> • Adicionando a Seção 3. Modelos de Projeto; • Adicionando a Seção 3.1 Diagrama de Classes. 	2.1
Guilherme Gabriel	12/04/2023	<ul style="list-style-type: none"> • Adicionando a Seção 3.2 Diagramas de Sequência. 	2.2
Guilherme Gabriel	13/04/2023	<ul style="list-style-type: none"> • Adicionando a Seção 3.3 Diagramas de Comunicação. 	2.3
Lucas Ângelo	15/04/2023	<ul style="list-style-type: none"> • Adicionando a Seção 3.4 Arquitetura. 	2.4

Lucas Ângelo	23/04/2023	<ul style="list-style-type: none"> • Atualizando versão do Diagrama de Classes. 	2.5
Guilherme Gabriel	27/04/2023	<ul style="list-style-type: none"> • Adicionando a Seção 3.5 Diagramas de Estados; • Adicionando a Seção 3.6 Diagrama de Componentes e Implantação. 	2.6
Lucas Ângelo	28/04/2023	<ul style="list-style-type: none"> • Corrigindo diagrama de pacotes; • Removendo palavra subseção do texto. 	2.7
Lucas Ângelo	29/04/2023	<ul style="list-style-type: none"> • Adicionando a Seção 5 Glossário e Modelos de Dados. 	2.8
Guilherme Gabriel	04/05/2023	<ul style="list-style-type: none"> • Adicionando a seção Diagrama de Componentes e Implantação; • Corrigindo erros textuais e em imagens apontados pelos professores. 	2.9
Lucas Ângelo	04/05/2023	<ul style="list-style-type: none"> • Atualizando numeração das figuras; • Corrigindo erros gramaticais. 	3.0
Guilherme Gabriel e Lucas Ângelo	06/05/2023	<ul style="list-style-type: none"> • Adicionando Seção 8.1 Cronograma. 	3.1
Lucas Ângelo	07/05/2023	<ul style="list-style-type: none"> • Adicionando Seção 6. Casos de Teste. 	3.2
Guilherme Gabriel	11/05/2023	<ul style="list-style-type: none"> • Adicionando Seção 6.1 Testes de Aceitação. 	3.3
Lucas Ângelo	12/05/2023	<ul style="list-style-type: none"> • Adicionando Seção 6.2 Testes de Integração. 	3.4
Guilherme Gabriel	19/05/2023	<ul style="list-style-type: none"> • Adicionando Seção 8.2 Processo de Implementação. 	3.5
Lucas Ângelo	23/05/2023	<ul style="list-style-type: none"> • Correção acrescentando um recuo ao início de todos os parágrafos, exceto o primeiro de cada seção. 	3.6
Lucas Ângelo	25/05/2023	<ul style="list-style-type: none"> • Atualizando nome da aplicação para GitGrade; • Corrigindo erros gramaticais. 	3.7
Guilherme Gabriel	25/05/2023	<ul style="list-style-type: none"> • Acrescentando informações na Seção 8.2 Processo de Implementação. 	3.8
Guilherme Gabriel e Lucas Ângelo	18/06/2023	<ul style="list-style-type: none"> • Adicionando testes de exceção, de aceitação e integração; • Correções de formatação 	4.0
Lucas Ângelo	28/06/2023	<ul style="list-style-type: none"> • Acrescentando benefícios do sistema na Seção 1. Introdução; • Adicionando Seção 7. Riscos. 	4.1

Guilherme Gabriel e Lucas Ângelo	25/06/2023	<ul style="list-style-type: none"> • Atualizando tarefas da <i>Sprint 1</i> da Seção 8.1 Cronograma; • Removendo menções ao uso de GraphQL. 	5.0
Lucas Ângelo	14/10/2023	<ul style="list-style-type: none"> • Atualizando o Diagrama de Entidade e Relacionamento da Seção 5 Glossário e Modelos de Dados. 	5.1
Lucas Ângelo	03/11/2023	<ul style="list-style-type: none"> • Atualizando o Diagrama de Caso de Uso da Seção 2.3.1; • Atualizando o Diagrama de Entidade e Relacionamento da Seção 5 Glossário e Modelos de Dados; • Atualizando o Diagrama de Classes de modelos do sistema da Seção 3.1 Diagrama de Classes; • Atualizando o Diagrama de Classes de serviços do sistema da Seção 3.1 Diagrama de Classes; • Atualizando o Diagrama de Classes de controladores do sistema da Seção 3.1 Diagrama de Classes. 	5.2
Lucas Ângelo	04/11/2023	<ul style="list-style-type: none"> • Atualizando o Diagramas de Comunicação (Figuras 19, 20, 21, 22, 23 e 24) da Seção 3.3 • Atualizando os Diagramas de Sequência (Figuras 14, 15, 16, 17 e 18) da Seção 3.2. • Adicionados parágrafos explicativos para cálculos e métricas na Seção 4. Projeto de Interface com Usuário. 	5.3
Lucas Ângelo	18/11/2023	<ul style="list-style-type: none"> • Adicionado Seção 9 Post-mortem. 	6.0
Guilherme Gabriel e Lucas Ângelo	26/11/2023	<ul style="list-style-type: none"> • Atualizando Seção 9 Post-mortem com explicações do que foi decidido não implementar; • Atualizando detalhes de saída dos Testes de Aceitação da Seção 6.1; • Adicionando testes de integração na Seção 6.2 Testes de Integração; • Revisão geral do documento. 	7.0

1. Introdução

Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o sistema GitGrade. 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 sistema proposto neste trabalho consiste em uma plataforma web de apoio às avaliações de trabalhos no GitHub¹ que será utilizado pelos professores das disciplinas de Trabalho Interdisciplinar. Nessa plataforma os professores poderão cadastrar métodos avaliativos para cada oferta de disciplina e visualizar as informações resultantes dos repositórios dos trabalhos conforme o método avaliativo selecionado. A necessidade de uma plataforma com essas funções origina-se da carência de uma aplicação que auxilie professores a avaliarem repositórios de código e os artefatos de documentação de trabalhos, por meio de filtros temporais e por integrantes.

O fluxo de funcionamento da plataforma inicia com o cadastro da oferta de uma disciplina em um semestre específico, com isso, o professor dessa disciplina deve selecionar quais são os repositórios dessa oferta. A partir disso, deve ser cadastrado um método avaliativo para essa disciplina, informando quais são os arquivos e seus respectivos diretórios onde devem estar criados e preenchidos. Por exemplo, na plataforma deve ser cadastrada a oferta da disciplina de “Trabalho Interdisciplinar 5: Aplicações Distribuídas 1º/2023” e seu método avaliativo contendo uma regra de consistência que exija a existência do arquivo “Documentacao/documento_de_arquitetura.md” contendo no mínimo 750 caracteres. A partir disso, todos os repositórios dessa oferta de disciplina verificam essa regra. Quando essa regra não for seguida por algum repositório de um trabalho com esse método avaliativo, é possível utilizar a opção de abrir uma *issue* padronizada no respectivo repositório.

Em relação ao controle temporal, também pode-se cadastrar *sprints* para ofertas da disciplina, o que possibilita avaliar contribuições de alunos de trabalhos em diferentes períodos temporais. Seguindo o mesmo exemplo da oferta da disciplina de “Trabalho Interdisciplinar 5: Aplicações Distribuídas 1º/2023”, podem ser cadastradas seis *sprints* para esse primeiro semestre de 2023, cada *sprint* com períodos pré-definidos de tempo, o que pode ser utilizado para filtrar contribuições de integrantes em trabalhos para cada uma dessas seis *sprints*. Por exemplo, para regra de consistência que exija a existência do arquivo “Documentacao/documento_de_arquitetura.md”, pode ser associada uma *sprint* que determina que, até a data final da *sprint*, esse documento deverá estar criado ou será aberta uma *issue* padronizada.

A partir da coleta dos dados dos repositórios do GitHub, resultados das validações das regras de consistências dos métodos avaliativos, contribuições de arquivos, linhas de código, *issues* e qualidade de código, a plataforma informa as contribuições no trabalho, por aluno e prazos de *sprints* para cada trabalho. Além disso, apresenta uma visualização de histórico de *commits* com filtro para alunos e *sprint* para cada repositório. Ademais, para cada repositório é apresentada a quantidade de *issues* abertas e fechadas por cada integrante de cada equipe, também sendo possível filtrar essa informação por *sprint*.

¹ <https://github.com/>

Esta plataforma conta com uma opção para efetuar uma inspeção da qualidade dos códigos por meio da ferramenta SonarQube² para cada repositório, essa inspeção calcula a qualidade do código do trabalho. Outrossim, para facilitar o uso por meio dos professores, o sistema de autenticação e autorização utiliza o OAuth do próprio GitHub. Dessa forma, os professores são auxiliados na avaliação tanto qualitativa quanto quantitativa das entregas dos artefatos, documentação e qualidade de código dos trabalhos desenvolvidos pelas equipes de alunos.

Diante disso, o GitGrade oferece uma série de benefícios aos docentes, especialmente aos professores responsáveis pela avaliação de trabalhos interdisciplinares no GitHub. A aplicação centraliza as informações relacionadas aos trabalhos acadêmicos em uma única plataforma, simplificando o acesso e a gestão desses dados.

Uma das principais vantagens do GitGrade é a possibilidade de estabelecer critérios de avaliação padronizados. Os professores podem cadastrar métodos avaliativos, definindo regras de consistência e requisitos específicos para os artefatos dos trabalhos. Isso facilita a comparação entre os alunos, promovendo uma avaliação mais objetiva e justa.

Outro aspecto do GitGrade é a capacidade de realizar análises temporais detalhadas. Através da criação de *sprints*, é possível avaliar as contribuições dos alunos ao longo de diferentes períodos. Essa funcionalidade permite identificar padrões de desempenho, acompanhar o progresso individual e estimular o crescimento contínuo dos alunos ao longo do tempo.

A visualização do histórico de *commits* por aluno e *sprint* é uma funcionalidade poderosa oferecida pelo GitGrade. A partir dela, os professores podem acompanhar de perto o desenvolvimento dos trabalhos, identificando as contribuições individuais e verificando a evolução dos projetos. Isso facilita a identificação de pontos fortes e áreas que necessitam de maior atenção, disponibilizando um feedback mais preciso e personalizado.

A integração do GitGrade com a ferramenta SonarQube para inspeção da qualidade do código é outro benefício significativo. Os professores podem avaliar automaticamente a qualidade do código dos trabalhos, identificando problemas e oportunidades de melhoria. Isso contribui para a formação de habilidades de programação mais sólidas e o desenvolvimento de códigos mais eficientes e legíveis.

Por fim, o GitGrade simplifica o processo de autenticação e autorização, utilizando o OAuth do próprio GitHub. Isso garante a segurança dos dados e proporciona uma experiência de uso intuitiva para os professores, reduzindo possíveis obstáculos e agilizando o acesso à plataforma.

2. Modelos de Usuário e Requisitos

Nesta seção são apresentados os modelos de usuário e requisitos do sistema. Mais especificamente, a Seção 2.1 descreve os atores do sistema, a Seção 2.2 discorre sobre o modelo de usuário, a Seção 2.3 sobre os casos de uso e histórias de usuário e, por fim, a Seção 2.4, apresenta o diagrama de sequência do sistema e o contrato de operações. Os diagramas apresentados nesta seção foram modelados por meio da Linguagem de Modelagem Unificada (UML, do inglês *Unified Modeling Language*).

² <https://www.sonarsource.com/products/sonarqube/>

2.1 Descrição de Atores

Sobre os atores que interagem com a plataforma, pode-se dividi-los em primários e secundários. Os primários são aqueles que interagem diretamente com a plataforma, enquanto os secundários são aqueles que impactam ou são impactados indiretamente pelas ações realizadas na plataforma. Portanto, pode-se definir que os atores da plataforma são os professores de disciplinas de trabalho interdisciplinar, os alunos das mesmas e o GitHub. A seguir são detalhados, individualmente, seu papel na plataforma.

O professor é o ator primário da plataforma, ou seja, o usuário que a utiliza diretamente. Ele é responsável por lecionar as disciplinas de trabalho interdisciplinar e, consequentemente, avaliar os repositórios dos alunos que cursam a disciplina. Seu objetivo principal na plataforma é acompanhar as entregas dos alunos.

O GitHub é um ator secundário do sistema. Ele é a plataforma de hospedagem de códigos que armazena os trabalhos dos alunos que cursam a disciplina. O próprio GitHub também é impactado pelo sistema, com a criação de *issues* a partir da detecção de erros nos repositórios dos alunos na plataforma.

O aluno é um ator secundário, impactado indiretamente pelas ações da plataforma. Eles cursam as disciplinas de trabalho interdisciplinar e realizam as entregas através do GitHub, sendo avaliados pelos professores. Além de alimentar os dados usados nas avaliações dentro da plataforma, ele recebe *feedbacks* através da criação de *issues* no GitHub.

2.2 Modelos de Usuários

Nesta seção são apresentados os modelos de usuários da plataforma. O modelo foi construído usando a estratégia de personas, no qual são especificados a biografia, personalidade, necessidades e frustrações. As personas são descrições de pessoas fictícias que representam o público alvo da plataforma. A Figura 1 apresenta a persona que representa o usuário ativo da plataforma, o professor. Enquanto isso, a Figura 2 apresenta a persona que representa os alunos, impactados pelo uso do sistema.

Vanessa



IDADE 35
FORMAÇÃO Mestrado
CARGO Professora
LOCALIZAÇÃO Belo Horizonte

Personalidade

Entusiasta Comunicativa

Bio

Residente de Belo Horizonte, possui um mestrado na área de engenharia de software. Há 4 anos faz parte do corpo docente de Engenharia de Software na PUC Minas, lecionando a disciplina de Trabalho Interdisciplinar IV.

Necessidades

- Precisa acompanhar e avaliar as entregas dos alunos na disciplina que leciona
- Melhores ferramentas para auxiliar no processo de aprendizagem dos seus alunos.

Frustrações

- O processo manual de avaliação das entregas é demorado.
- Falta de tempo hábil para formular retornos mais específicos para cada grupo sobre suas entregas.

Figura 1. Persona que representa um professor



Figura 2. Persona que representa um aluno

2.3 Modelo de Casos de Uso e Histórias de Usuários

Esta seção descreve os casos de uso e as histórias de usuário que contemplam as possíveis ações dos usuários da plataforma desenvolvida neste projeto, apresentados respectivamente nas seções 2.3.1 e 2.3.2. Para descrever os casos de uso, foi criado o Diagrama de Caso de Uso, o qual é responsável por descrever as interações entre a plataforma deste projeto e seus usuários ou outros sistemas, ele mostra as ações que um usuário ou sistema pode realizar em relação à plataforma em questão. As histórias de usuário tem objetivo de apresentar as necessidades do usuário em relação à plataforma de uma maneira clara e concisa para facilitar o entendimento das funcionalidades.

2.3.1 Diagrama de Casos de Uso

A Figura 3 ilustra o Diagrama dos Casos de Uso do sistema. Nesse diagrama é possível observar 3 atores da plataforma, sendo o ator primário Professor e os atores secundários GitHub e Aluno. O ator Professor representa os professores que acessam o sistema para efetuar as operações básicas de cadastro, atualização, deleção e visualização das funcionalidades de trabalhos, métodos avaliativos, regras de consistência, análise estática de código e *issues* padronizadas. Já o ator secundário GitHub é utilizado pela execução de tarefas cronometradas de busca e atualização da base de dados do sistema por meio da Interface de Programação de Aplicação (API, do inglês *Application Programming Interface*) do GitHub, também é utilizado para autenticação de usuários. Por fim, o ator Aluno é utilizado para análises de contribuições individuais nos repositórios de trabalhos.

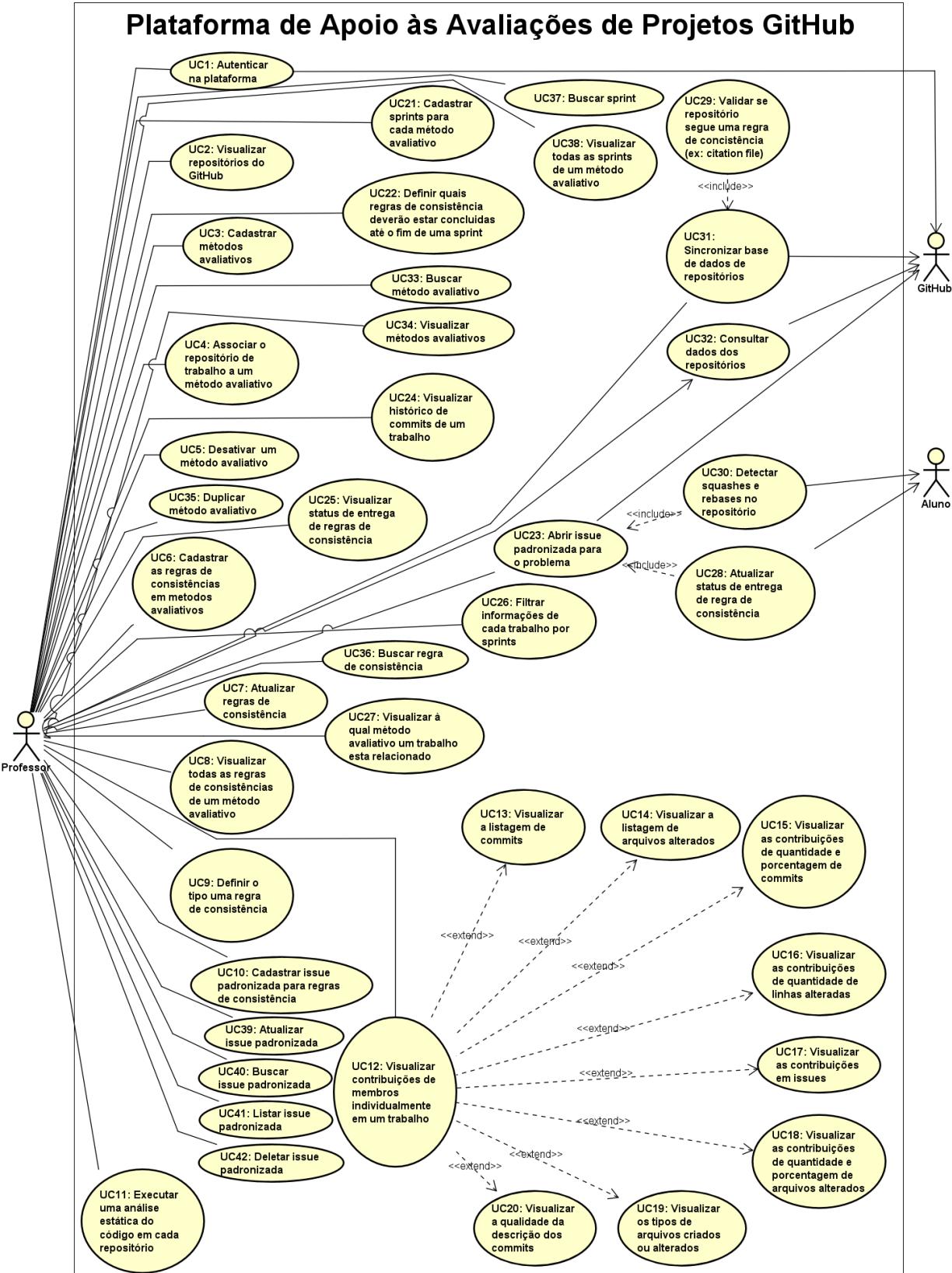


Figura 3. Diagrama de Caso de Uso

2.3.2 Histórias de Usuários

As histórias de usuário (US, do inglês *User Stories*) são uma forma de representar os requisitos do cliente. Trata-se de uma descrição simples, que informa quem realizará uma determinada ação, qual será essa ação, e a razão que ela é realizada. A seguir, são descritas as histórias de usuários levantadas para a plataforma, identificadas pela sigla US e uma numeração.

US01. Como professor, desejo me autenticar usando a conta do GitHub, para usar as funcionalidades do sistema (relativo ao UC01);

US02. Como professor, desejo ver os repositórios que sou responsável, para poder avaliá-los (relativo ao UC02);

US03. Como professor, desejo criar um método avaliativo, para vincular aos repositórios que avaliarei (relativo ao UC03);

US04. Como professor, desejo ver os métodos avaliativos do sistema, para usar nos repositórios que devo avaliar (relativo ao UC33 e UC34);

US05. Como professor, desejo vincular um repositório, que sou responsável, a um método avaliativo, para o repositório ser avaliado de acordo com aquele método (relativo ao UC04);

US06. Como professor, desejo inativar um método avaliativo obsoleto, para que ele não possa mais ser vinculado a novos repositórios (relativo ao UC05);

US07. Como professor, desejo criar uma regra de consistência em um método avaliativo, para que ela seja validada nos repositórios vinculados ao método (relativo ao UC06, UC09, UC22 e UC29);

US08. Como professor, desejo editar as regras de consistência, para que eu possa corrigir informações anteriormente cadastradas (relativo ao UC07);

US09. Como professor, desejo ver todas as regras de consistências cadastradas para um método avaliativo, para que eu possa saber como o conjunto de repositórios vinculados será avaliado (relativo ao UC08 e UC36);

US10. Como professor, desejo criar um *template* de *issue* padronizada em um método avaliativo, para ser vinculado nas regras de consistência que devem disparar a *issue* no repositório vinculado, caso ela não seja cumprida (relativo ao UC10 e UC23);

US11. Como professor, desejo ver as *issues* padronizadas criadas para um método avaliativo, para saber quais posso vincular às regras de consistência do método (relativo ao UC40 e UC41);

US12. Como professor, desejo editar a *issue* padronizada criada no método avaliativo, para que eu possa corrigir informações anteriormente cadastradas (relativo ao UC39);

US13. Como professor, desejo deletar a *issue* padronizada criada no método avaliativo, para não ser mais possível o vincular a uma regra de consistência (relativo ao UC42).

US14. Como professor, desejo definir as *sprints* de um método avaliativo, para agrupar as entregas dos repositórios vinculados (relativo ao UC21);

US15. Como professor, desejo ver as *sprints* de um método avaliativo, para saber quando são as entregas que os trabalhos devem realizar (relativo ao UC37 e UC38);

US16. Como professor, desejo duplicar um método avaliativo, para que eu possa reutilizar suas configurações para outra turma (relativo ao UC35);

US17. Como professor, desejo sincronizar as informações do repositório com o GitHub, para poder avaliar os alunos com as informações atualizadas (relativo ao UC28, UC30, UC31 e UC32);

US18. Como professor, desejo ver qual método avaliativo um repositório pertence, para saber quais regras se aplicam a avaliação dele (relativo ao UC27);

US19. Como professor, desejo conferir os indicadores de contribuição (quantidade e qualidade da descrição de *commits*, linhas alteradas, quantidade e tipos de arquivos alterados e *issues* abertas e fechadas) de um repositório, para poder avaliar o empenho geral do grupo (relativo ao UC15, UC16, UC17, UC18, UC19 e UC20);

US20. Como professor, desejo filtrar os indicadores de contribuição por contribuidor, para poder avaliar o empenho geral do integrante do grupo (relativo ao UC12);

US21. Como professor, desejo filtrar os indicadores de contribuição por *sprint*, para poder avaliar o empenho naquela entrega (relativo ao UC26);

US22. Como professor, desejo ver quais regras de consistência foram cumpridas em um determinado repositório, para usar essa informação para o cálculo da nota do grupo (relativo ao UC25);

US23. Como professor, desejo ver quais os *commits* foram realizados no repositório pelos integrantes, para ter uma visão mais específica do tipo contribuição que os alunos estão fazendo (relativo ao UC13 e UC24);

US24. Como professor, desejo filtrar a visualização de *commits* para mostrar apenas os afetados for *force push*, para saber se os integrantes do grupo estão fazendo *squashes* e *rebases* no repositório. (relativo ao UC13, UC24 e UC30);

US25. Como professor, desejo ver quais arquivos foram alterados pelos integrantes, para ter uma visão mais específica do tipo contribuição que os alunos estão fazendo (relativo ao UC14);

US26. Como professor, desejo disparar a execução de uma ferramenta de análise estática de código (SonarQube) no código do repositório, para avaliar a qualidade do código que está sendo produzidos pelos alunos (relativo ao UC11);

US27. Como professor, desejo ver os resultados da execução de uma ferramenta de análise estática de código (SonarQube) no código do repositório, para avaliar a qualidade do código que está sendo produzidos pelos alunos (relativo ao UC11);

2.4 Diagrama de Sequência do Sistema e Contrato de Operações

Nesta seção, são apresentados os diagramas de sequência do sistema (DSS) com seus respectivos contratos de operações, sendo baseados nos casos de uso apresentados na Seção 2.3.1. Esses diagramas têm como finalidade descrever os fluxos de interação que ocorrem entre os usuários e o sistema desenvolvido.

A Figura 4 apresenta o DSS 1 relacionado ao fluxo de autenticar na plataforma. Nesse fluxo, o usuário Professor envia uma mensagem síncrona ao sistema com o objetivo de se autenticar na plataforma utilizando suas credenciais do GitHub. Ademais, esse fluxo também contata o sistema GitHub com intuito de validar as credenciais informadas. Por fim, o usuário recebe sua autorização para prosseguir utilizando a plataforma. Esse diagrama contempla o caso de uso UC1. Além disso, o contrato de operação desse diagrama é detalhado pela Tabela 1.

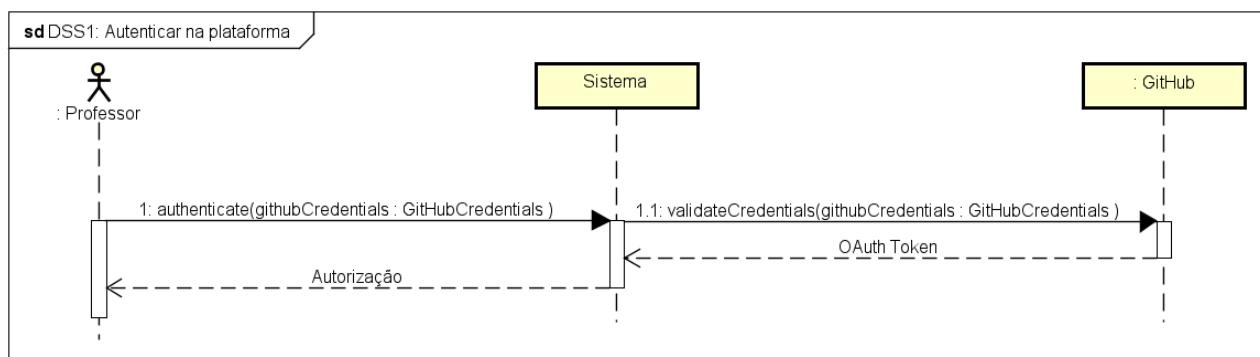


Figura 4. DSS 1: Autenticar na plataforma

Contrato	Autenticar na plataforma.
Operação	authenticate(githubCredentials : GitHubCredentials)
Referências cruzadas	UC1: Autenticar na plataforma.
Pré-condições	Professor abre o sistema web sem estar autenticado.
Pós-condições	Iniciada uma sessão no sistema, o usuário está devidamente autenticado com uma autorização.

Tabela 1. Contrato de operação para autenticar

A Figura 5 apresenta o DSS 2 relacionado ao fluxo de gerenciar métodos avaliativos. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de criar, atualizar, buscar, listar, desativar, duplicar ou adicionar um repositório a um método avaliativo. Dessa forma, o Professor consegue gerenciar os métodos avaliativos na plataforma. Esse diagrama contempla os casos de uso UC3, UC4, UC5, UC6, UC21, UC33, UC34 e UC35. Além disso, os contratos de operações desse diagrama são detalhados pelas Tabelas 2, 3, 4, 5, 6, 7 e 8.

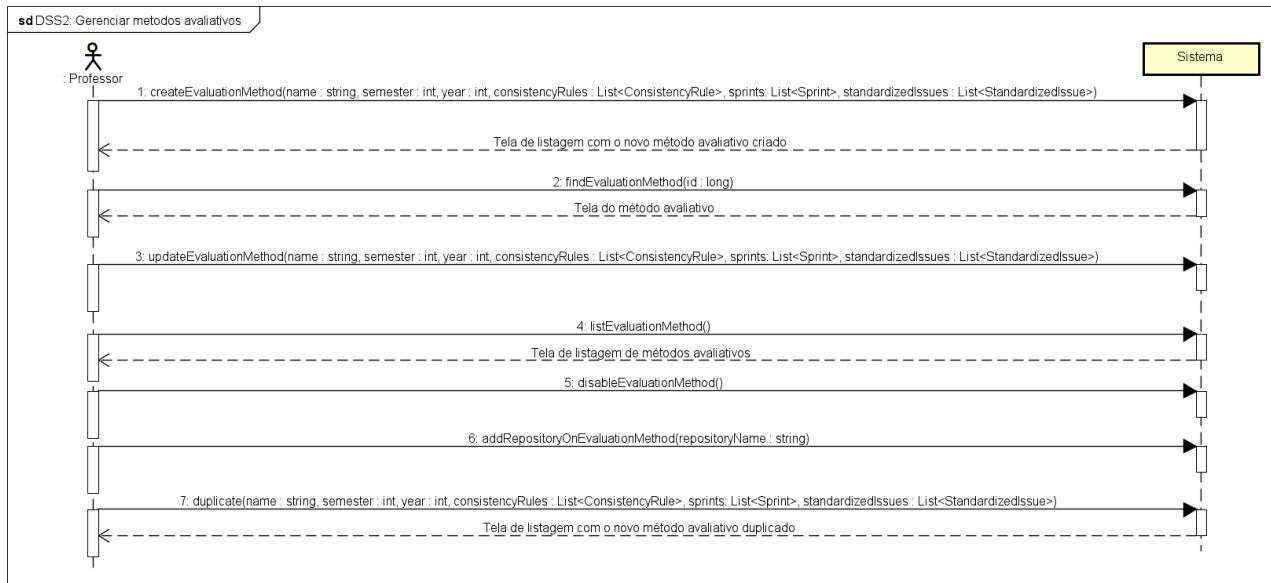


Figura 5. DSS 2: Gerenciar método avaliativos

Contrato	Criar um método avaliativo.
Operação	createEvaluationMethod(name : string, semester : int, year : int, consistencyRules : List<ConsistencyRule>, sprints: List<Sprint>, standardizedIssues : List<StandardizedIssue>)
Referências cruzadas	UC3: Cadastrar métodos avaliativos; UC6: Cadastrar as regras de consistências em métodos avaliativos; UC21: Cadastrar sprints para cada método avaliativo.
Pré-condições	Usuário estar autenticado.
Pós-condições	O método avaliativo ser criado e a listagem de métodos atualizar.

Tabela 2. Contrato de operação para criar um método avaliativo

Contrato	Buscar um método avaliativo.
Operação	findEvaluationMethod(id : long)
Referências cruzadas	UC33: Buscar método avaliativo.
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	O método avaliativo deve ser apresentado.

Tabela 3. Contrato de operação para buscar um método avaliativo.

Contrato	Atualizar um método avaliativo.
Operação	updateEvaluationMethod(name : string, semester : int, year : int, consistencyRules : List<ConsistencyRule>, sprints: List<Sprint>, standardizedIssues : List<StandardizedIssue>)
Referências cruzadas	UC6: Cadastrar as regras de consistências em métodos avaliativos; UC21: Cadastrar <i>sprints</i> para cada método avaliativo.
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	O método avaliativo deve ser atualizado.

Tabela 4. Contrato de operação para atualizar um método avaliativo

Contrato	Visualizar os métodos avaliativos.
Operação	listEvaluationMethod()
Referências cruzadas	UC34: Visualizar métodos avaliativos.
Pré-condições	Usuário estar autenticado.
Pós-condições	Uma listagem dos métodos avaliativos ser apresentada.

Tabela 5. Contrato de operação para visualizar os métodos avaliativos

Contrato	Desativar um método avaliativo.
Operação	disableEvaluationMethod()
Referências cruzadas	UC5: Desativar um método avaliativo.
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	O método avaliativo deve ser desativado com deleção lógica.

Tabela 6. Contrato de operação para desativar um método avaliativo

Contrato	Adicionar um repositório a um método avaliativo.
Operação	addRepositoryOnEvaluationMethod(repositoryName : string)
Referências cruzadas	UC4: Associar o repositório de trabalho a um método avaliativo.
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	O repositório deve ser associado ao método avaliativo.

Tabela 7. Contrato de operação para adicionar um repositório a um método avaliativo

Contrato	Duplicar um método avaliativo.
Operação	duplicate(name : string, semester : int, year : int, consistencyRules : List<ConsistencyRule>, sprints: List<Sprint>, standardizedIssues : List<StandardizedIssue>)
Referências cruzadas	UC35: Duplicar um método avaliativo; UC6: Cadastrar as regras de consistências em métodos avaliativos; UC21: Cadastrar <i>sprints</i> para cada método avaliativo.
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	O método avaliativo deve duplicado juntamente com seus objetos associados.

Tabela 8. Contrato de operação para duplicar um método avaliativo

A Figura 6 apresenta o DSS 3 relacionado ao fluxo de gerenciar regras de consistência. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de criar, atualizar, buscar e listar regras de consistência. Dessa forma, o Professor consegue gerenciar as regras de consistência dos métodos avaliativos na plataforma. Esse diagrama contempla os casos de uso UC6, UC7, UC8, UC9, UC22 e UC36. Além disso, os contratos de operações desse diagrama são detalhados pelas Tabelas 9, 10, 11 e 12.

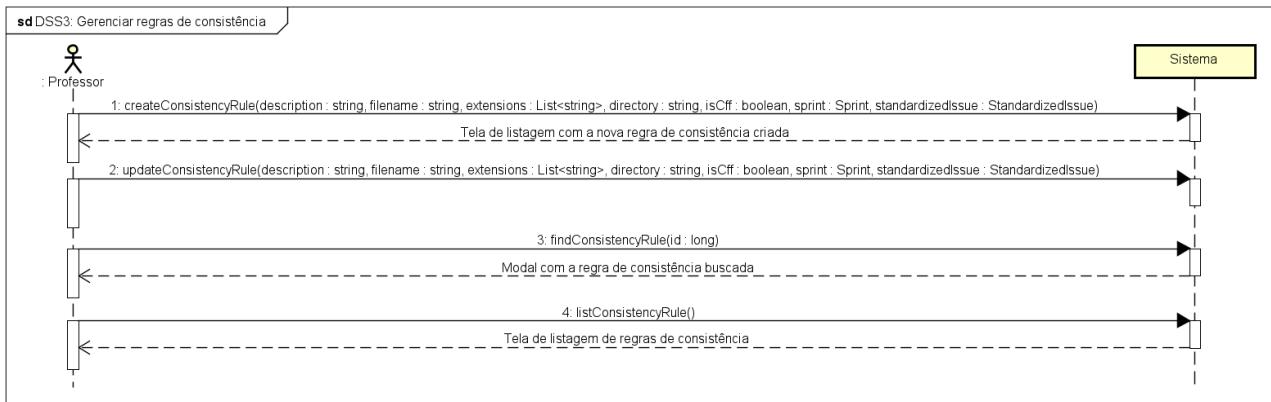


Figura 6. DSS 3: Gerenciar regras de consistência

Contrato	Criar uma regra de consistência.
Operação	createConsistencyRule(description : string, filename : string, extensions : List<string>, directory : string, isCff : boolean, sprint : Sprint, standardizedIssue : StandardizedIssue)
Referências cruzadas	UC6: Cadastrar as regras de consistências em métodos avaliativos; UC9: Definir o tipo uma regra de consistência; UC22: Definir quais regras de consistência deverão estar concluídas até o fim de uma sprint.
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	A regra de consistência ser criada e a listagem de regras do método avaliativo correspondente atualizar.

Tabela 9. Contrato de operação para criar uma regra de consistência

Contrato	Atualizar uma regra de consistência.
Operação	updateConsistencyRule(description : string, filename : string, extensions : List<string>, directory : string, isCff : boolean, sprint : Sprint, standardizedIssue : StandardizedIssue)
Referências cruzadas	UC7: Atualizar regras de consistência.
Pré-condições	Usuário estar autenticado e uma regra de consistência estar cadastrada.
Pós-condições	A regra de consistência ser atualizada e a listagem de regras do método avaliativo correspondente atualizar.

Tabela 10. Contrato de operação para atualizar uma regra de consistência

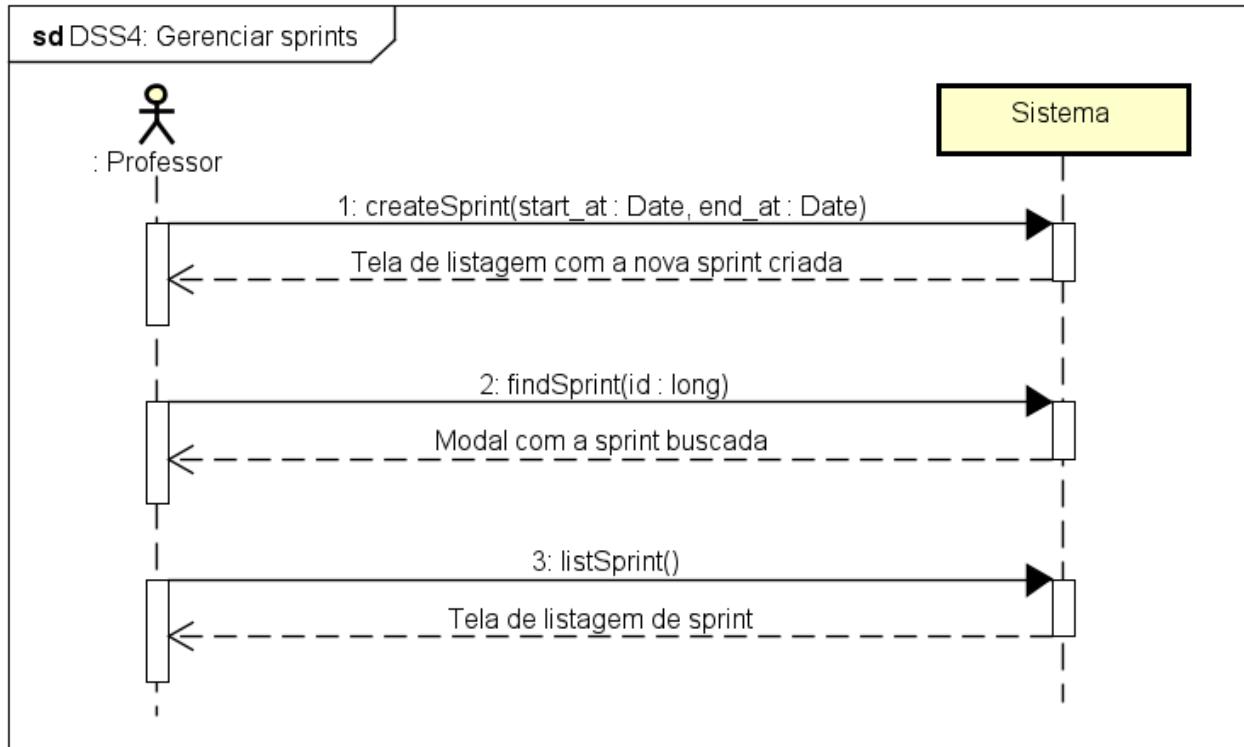
Contrato	Buscar uma regra de consistência.
Operação	findConsistencyRule(id : long)
Referências cruzadas	UC36: Buscar regra de consistência.
Pré-condições	Usuário estar autenticado e uma regra de consistência estar cadastrada.
Pós-condições	A regra de consistência deve ser apresentada.

Tabela 11. Contrato de operação para buscar uma regra de consistência

Contrato	Visualizar as regras de consistência.
Operação	listConsistencyRule()
Referências cruzadas	UC8: Visualizar todas as regras de consistências de um método avaliativo.
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	Uma listagem das regras de consistência do método avaliativo correspondente ser apresentada.

Tabela 12. Contrato de operação para visualizar as regras de consistência

A Figura 7 apresenta o DSS 4 relacionado ao fluxo de gerenciar *sprints*. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de criar, buscar e listar *sprints*. Dessa forma, o Professor consegue gerenciar as *sprints* dos métodos avaliativos na plataforma. Esse diagrama contempla os casos de uso UC21, UC22, UC37 e UC38. Além disso, os contratos de operações desse diagrama são detalhados pelas Tabelas 13, 14, e 15.

Figura 7. DSS 4: Gerenciar *sprints*

Contrato	Criar <i>sprint</i> .
Operação	createSprint(start_at : Date, end_at : Date)
Referências cruzadas	UC21: Cadastrar <i>sprints</i> para cada método avaliativo.
Pré-condições	Usuário estar autenticado e um método avaliativo estar criado.
Pós-condições	A <i>sprint</i> ser criada e a listagem de <i>sprints</i> do método avaliativo correspondente atualizar.

Tabela 13. Contrato de operação para criar uma *sprint*

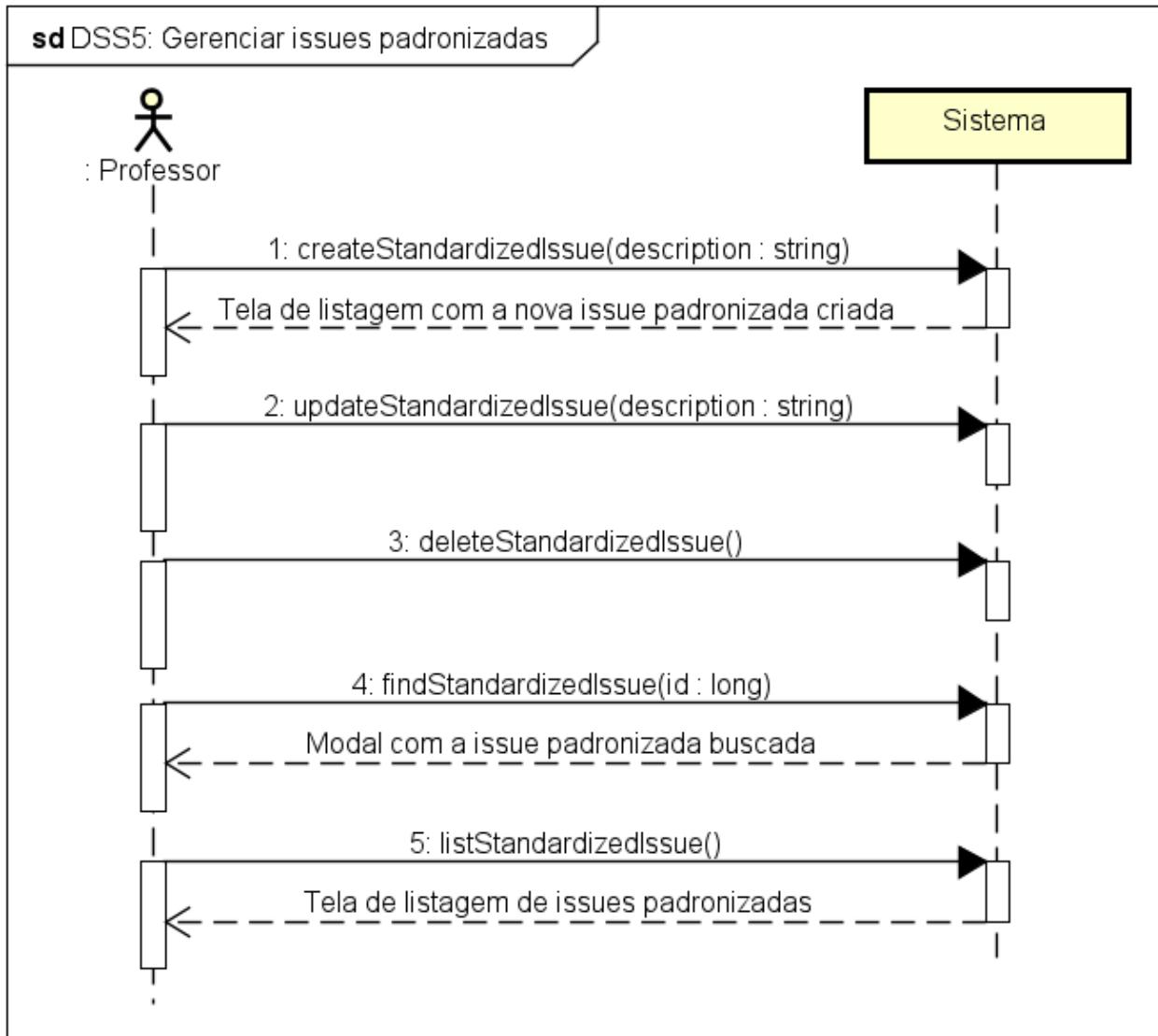
Contrato	Buscar uma <i>sprint</i> .
Operação	findSprint(id : long)
Referências cruzadas	UC37: Buscar <i>sprint</i> .
Pré-condições	Usuário estar autenticado e uma <i>sprint</i> estar criada.
Pós-condições	A <i>sprint</i> deve ser apresentada.

Tabela 14. Contrato de operação para buscar uma *sprint*

Contrato	Visualizar as <i>sprints</i> .
Operação	listSprint()
Referências cruzadas	UC38: Visualizar todas as <i>sprints</i> de um método avaliativo
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	Uma listagem das <i>sprints</i> do método avaliativo correspondente ser apresentada.

Tabela 15. Contrato de operação para visualizar as *sprints*

A Figura 8 apresenta o DSS 5 relacionado ao fluxo de gerenciar *issues* padronizadas. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de criar, atualizar, deletar, buscar e listar *issues* padronizadas. Dessa forma, o Professor consegue gerenciar as *issues* padronizadas dos métodos avaliativos na plataforma. Esse diagrama contempla os casos de uso UC10, UC39, UC40, UC41 e UC42. Além disso, os contratos de operações desse diagrama são detalhados pelas Tabelas 16, 17, 18, 19 e 20.

Figura 8. DSS 5: Gerenciar *issues* padronizadas

Contrato	Criar <i>issue</i> padronizada.
Operação	createStandardizedIssue(description : string)
Referências cruzadas	UC10: Cadastrar <i>issue</i> padronizada para regras de consistência.
Pré-condições	Usuário estar autenticado e possuir regra de consistência criada
Pós-condições	A <i>issue</i> padronizada ser criada e a listagem de <i>issues</i> padronizadas do método avaliativo correspondente atualizar.

Tabela 16. Contrato de operação para criar *issue* padronizada

Contrato	Atualizar <i>issue</i> padronizada.
Operação	updateStandardizedIssue(description : string)
Referências cruzadas	UC39: Atualizar <i>issue</i> padronizada.
Pré-condições	Usuário estar autenticado e possuir uma <i>issue</i> padronizada estar criada.
Pós-condições	A <i>issue</i> padronizada ser atualizada e a listagem de <i>issues</i> padronizadas do método avaliativo correspondente atualizar.

Tabela 17. Contrato de operação para atualizar *issue* padronizada

Contrato	Deletar <i>issue</i> padronizada.
Operação	deleteStandardizedIssue()
Referências cruzadas	UC42: Deletar <i>issue</i> padronizada.
Pré-condições	Usuário estar autenticado e possuir uma <i>issue</i> padronizada estar criada.
Pós-condições	A <i>issue</i> padronizada deve ser deletada.

Tabela 18. Contrato de operação para deletar *issue* padronizada

Contrato	Buscar <i>issue</i> padronizada.
Operação	findStandardizedIssue(id : long)
Referências cruzadas	UC40: Buscar <i>issue</i> padronizada.
Pré-condições	Usuário estar autenticado e possuir uma <i>issue</i> padronizada estar criada.
Pós-condições	A <i>issue</i> padronizada deve ser apresentada.

Tabela 19. Contrato de operação para buscar *issue* padronizada

Contrato	Listar <i>issues</i> padronizadas.
Operação	listStandardizedIssue()
Referências cruzadas	UC41: Listar <i>issue</i> padronizada.
Pré-condições	Usuário estar autenticado e um método avaliativo estar cadastrado.
Pós-condições	Uma listagem das <i>issues</i> padronizadas do método avaliativo correspondente ser apresentada.

Tabela 20. Contrato de operação para listar *issues* padronizadas

A Figura 9 apresenta o DSS 6 relacionado ao fluxo de gerenciar repositórios de trabalhos. Nesse fluxo, o usuário Professor envia mensagens síncronas e assíncronas ao sistema com o objetivo de sincronizar, buscar, listar, atualizar, calcular métricas e qualidade do código, buscar histórico de *commits* e consistência das entregas de um repositório de trabalho. Dessa forma, o Professor consegue gerenciar os repositórios de trabalhos. Esse diagrama contempla os casos de uso UC2, UC11, UC12, UC24, UC25, UC26, UC29, UC31 e UC32. Além disso, os contratos de operações desse diagrama são detalhados pelas Tabela 21, 22, 23, 24, 25, 26 e 27.

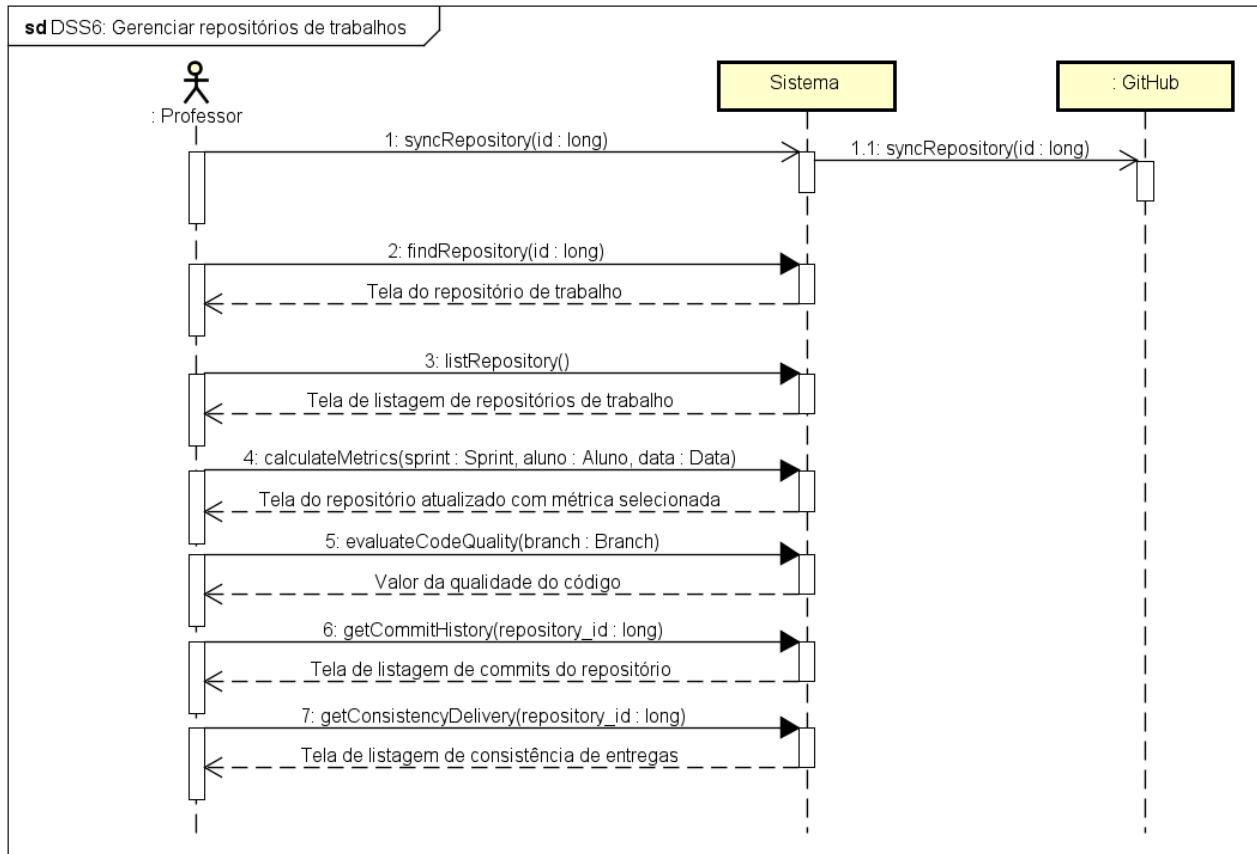


Figura 9. DSS 6: Gerenciar repositórios de trabalhos

Contrato	Sincronizar repositório de trabalho.
Operação	syncRepository(id : long)
Referências cruzadas	UC31: Sincronizar base de dados de repositórios.
Pré-condições	Usuário estar autenticado e possuir um repositório sincronizado.
Pós-condições	O repositório de trabalho ser sincronizado com a base de dados proveniente do GitHub.

Tabela 21. Contrato de operação para sincronizar repositório de trabalho

Contrato	Buscar repositório de trabalho.
Operação	findRepository(id : long)
Referências cruzadas	UC32: Consultar dados dos repositórios.
Pré-condições	Usuário estar autenticado e possuir um repositório sincronizado.
Pós-condições	O repositório de trabalho ser apresentado.

Tabela 22. Contrato de operação para buscar repositório de trabalho

Contrato	Listar repositórios de trabalhos.
Operação	listRepository()
Referências cruzadas	UC2: Visualizar repositórios do GitHub.
Pré-condições	Usuário estar autenticado e possuir um repositório sincronizado.

Pós-condições	A listagem de repositórios de trabalhos ser apresentada.
----------------------	--

Tabela 23. Contrato de operação para listar repositórios de trabalhos

Contrato	Calcular métricas de um repositório de trabalho.
Operação	calculateMetrics(sprint : Sprint, aluno : Aluno, data : Data)
Referências cruzadas	UC12: Visualizar contribuições de membros individualmente em um trabalho; UC26: Filtrar informações de cada trabalho por <i>sprints</i> .
Pré-condições	Usuário estar autenticado e possuir um repositório sincronizado.
Pós-condições	As métricas do repositório de trabalho correspondente serem calculadas.

Tabela 24. Contrato de operação para calcular métricas de um repositório de trabalho

Contrato	Calcular qualidade de código de um repositório de trabalho.
Operação	evaluateCodeQuality(branch : Branch)
Referências cruzadas	UC11: Executar uma análise estática do código em cada repositório.
Pré-condições	Usuário estar autenticado e possuir um repositório sincronizado.
Pós-condições	A qualidade de código do repositório de trabalho correspondente ser calculada.

Tabela 25. Contrato de operação para calcular qualidade de código de um repositório de trabalho

Contrato	Buscar histórico de <i>commits</i> de um repositório de trabalho.
Operação	getCommitHistory(repository_id : long)
Referências cruzadas	UC24: Visualizar histórico de commits de um trabalho.
Pré-condições	Usuário estar autenticado e possuir um repositório sincronizado.
Pós-condições	O histórico de <i>commits</i> do repositório de trabalho correspondente ser apresentada.

Tabela 26. Contrato de operação para buscar histórico de commits de um repositório de trabalho

Contrato	Buscar consistência de entregas um repositório de trabalho.
Operação	getConsistencyDelivery(repository_id : long)
Referências cruzadas	UC25: Visualizar status de entrega de regras de consistência; UC29: Validar se repositório segue uma regra de consistência (ex: citation file).
Pré-condições	Usuário estar autenticado e possuir um repositório sincronizado.
Pós-condições	A consistência de entregas baseado nas regras de consistência do repositório de trabalho correspondente ser apresentada.

Tabela 27. Contrato de operação para buscar consistência de entregas um repositório de trabalho

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 Classes

Nesta seção, é apresentado o diagrama de classes que modela objetos e suas relações e seus respectivos relacionamentos utilizados para tratar as informações da plataforma. O propósito desta seção consiste em apresentar todos os modelos de dados implementados, com o intuito de fornecer uma visão geral da arquitetura das classes que compõem o sistema e assegurar o funcionamento correto da plataforma.

A Figura 10 apresenta o diagrama de classes de modelos do sistema, o qual contém as principais classes *Repository*, *Branch*, *Commit*, *File*, *CodeQuality*, *EvaluationMethod*, *ConsistencyRule*, *ConsistencyDelivery*, *Sprint*, *StandardizedIssue*, *User*, *Contributor*, *Issue* e *PullRequest* responsáveis por armazenar e tratar os dados necessários inseridos na plataforma pelos professores e pelo GitHub. Além disso, contém também as classes secundárias *repository_has_contributor*, *issue_has_assigne_contributor* e *pullrequest_has_assigne_contributor* necessárias para tratar relações entre as respectivas classes principais. Todas essas classes são tratadas na plataforma durante a execução do sistema, utilizadas para permitir o tratamento das informações das interfaces internas da plataforma referentes as funcionalidades dos professores e interfaces externas da plataforma necessárias para sincronização de dados com o GitHub.

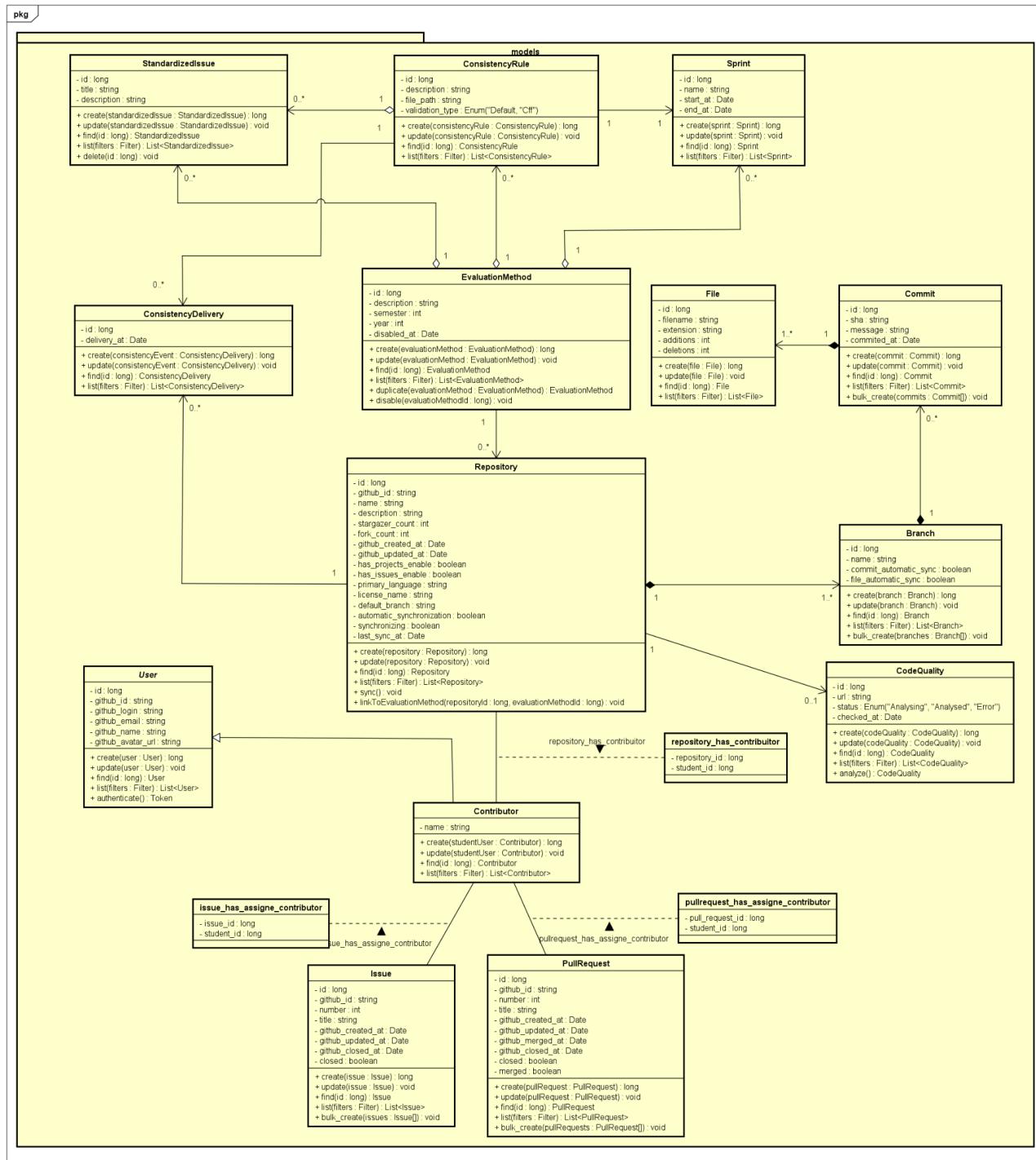


Figura 10. Diagrama de Classes de modelos do sistema

A Figura 11 apresenta o diagrama de classes de serviços do sistema, o qual contém as classes *AuthService*, *RepositoryService*, *EvaluationMethodService*, *ConsistencyRuleService*, *StandardizedIssueService* e *SprintService* responsáveis por manipular as funções das classes de modelo do sistema. Esse diagrama representa as classes de *services* invocadas pelas classes de

controllers, esses serviços manipulam diretamente os dados de modelos internos da plataforma, sem envolver informações de classes do sistema externo GitHub.

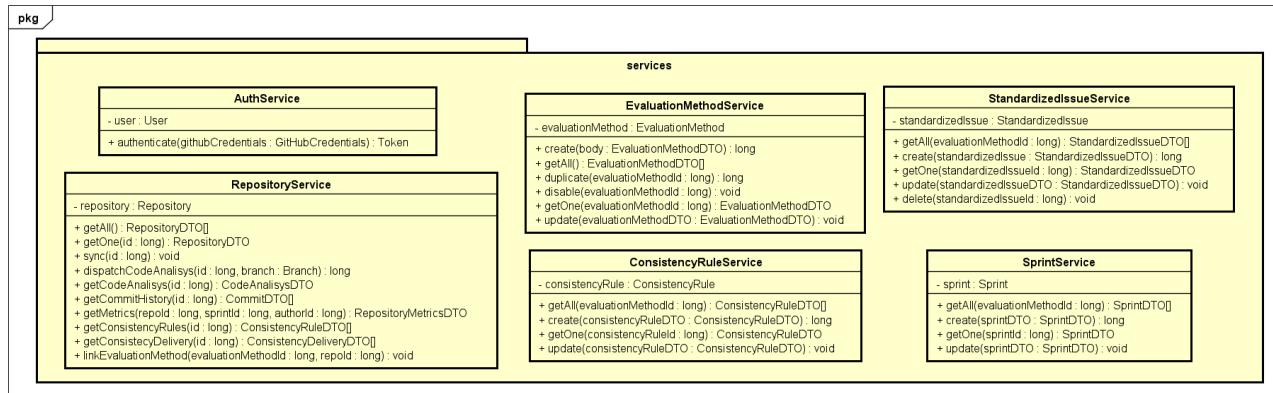


Figura 11. Diagrama de Classes de serviços do sistema

A Figura 12 apresenta o diagrama de classes de controladores do sistema, o qual contém as classes *AuthController*, *RepositoryController*, *EvaluationMethodController*, *ConsistencyRuleController*, *StandardizedIssueController* e *SprintController*. Elas são responsáveis por receber e tratar os dados de requisições do sistema. Esse diagrama representa as classes de *controllers* invocadas por chamadas da API, tratando as mensagens e encaminhando para o *service* correspondente, efetuando também o tratamento do retorno das informações.

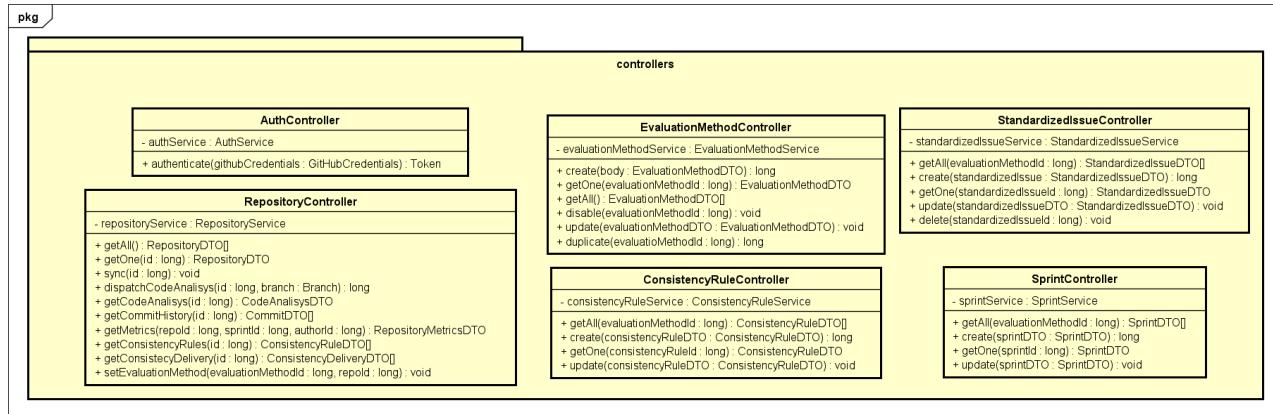


Figura 12. Diagrama de Classes de controladores do sistema

3.2 Diagramas de Sequência

Nesta seção, são apresentados diagramas de sequência referentes aos casos de uso do sistema implementado. Esses diagramas demonstram o fluxo entre os componentes para cada cenário de uso, dessa forma, eles apresentam como funcionam os fluxos de chamadas entre as entidades que participam de uma interação. As entidades desses diagramas foram apresentadas no diagrama de classes das Figura 10, Figura 11 e Figura 12.

A Figura 13 representa o diagrama de sequência responsável pelo fluxo de autenticação na plataforma. Nesse fluxo, o usuário Professor envia uma mensagem síncrona ao sistema com o objetivo de se autenticar na plataforma utilizando suas credenciais do GitHub. A partir disso, a mensagem chega ao componente de execução *AuthController* que irá tratar os dados da requisição, com isso, enviar para o sistema externo GitHub com intuito de validar as credenciais informadas. Ao sucesso da validação, a autorização é gerada pelo componente de execução *AuthService* e o usuário pode prosseguir utilizando a plataforma. Esse diagrama contempla o caso de uso UC1.

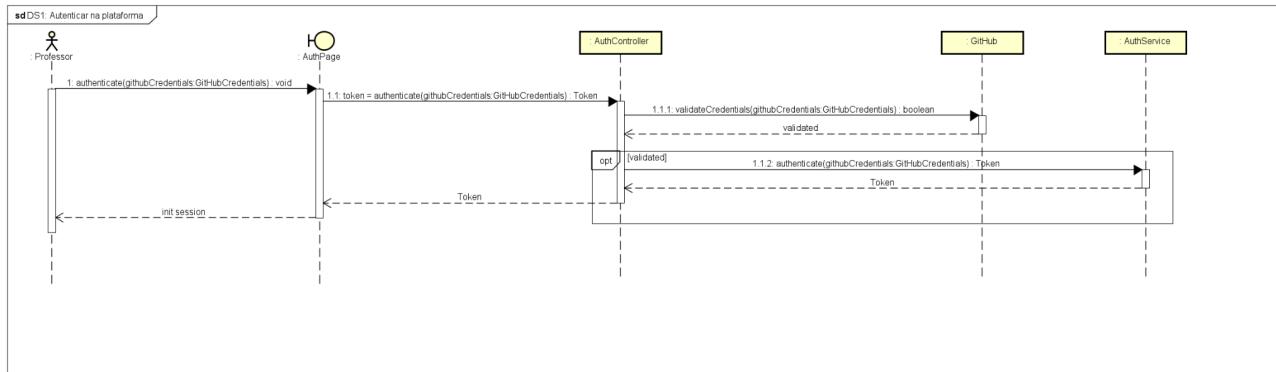


Figura 13. DS 1: Autenticar na plataforma

A Figura 14 representa o diagrama de sequência responsável pelo fluxo de gerenciar métodos avaliativos. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de criar, atualizar, buscar, listar, desativar, duplicar ou adicionar um repositório a um método avaliativo. A partir disso, as mensagens chegam ao componente de execução *EvaluationMethodController* que trata os dados da requisição e, com isso, enviar para o componente de execução *EvaluationMethodService* que manipula a entidade *EvaluationMethod* dentro do sistema conforme o objetivo requerido pelo usuário. Esse diagrama contempla os casos de uso UC3, UC4, UC5, UC6, UC21, UC33, UC34 e UC35.



Figura 14. DS 2: Gerenciar métodos avaliativos

A Figura 15 representa o diagrama de sequência responsável pelo fluxo de gerenciar regras de consistência. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de criar, atualizar, buscar e listar regras de consistência. A partir disso, as mensagens chegam ao componente de execução *ConsistencyRuleController* que trata os dados da requisição e, com isso, enviar para o componente de execução *ConsistencyRuleService* que manipula a entidade *ConsistencyRule* dentro do sistema conforme o objetivo requerido pelo usuário. Esse diagrama contempla os casos de uso UC6, UC7, UC8, UC9, UC22 e UC36.

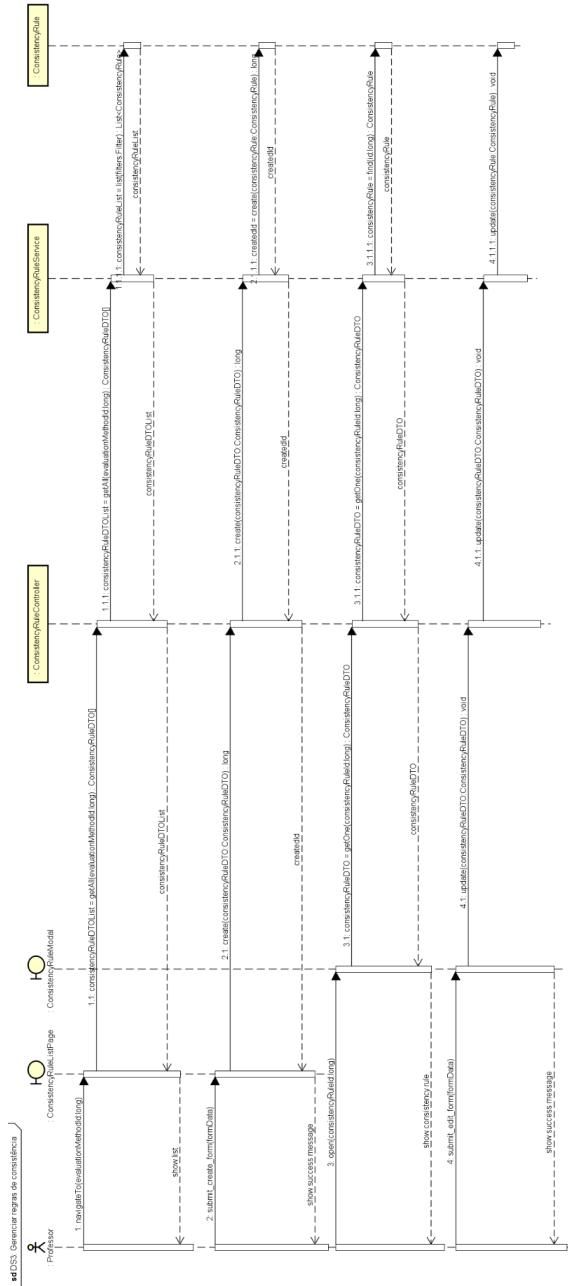


Figura 15. DS 3: Gerenciar regras de consistência

A Figura 16 representa o diagrama de sequência responsável pelo fluxo de gerenciar *sprints*. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de criar, buscar e listar *sprints*. A partir disso, as mensagens chegam ao componente de execução *SprintController* que trata os dados da requisição e, com isso, enviar para o componente de execução *SprintService* que manipula a entidade *Sprint* dentro do sistema conforme o objetivo requerido pelo usuário. Esse diagrama contempla os caso de uso UC21, UC22, UC37 e UC38.

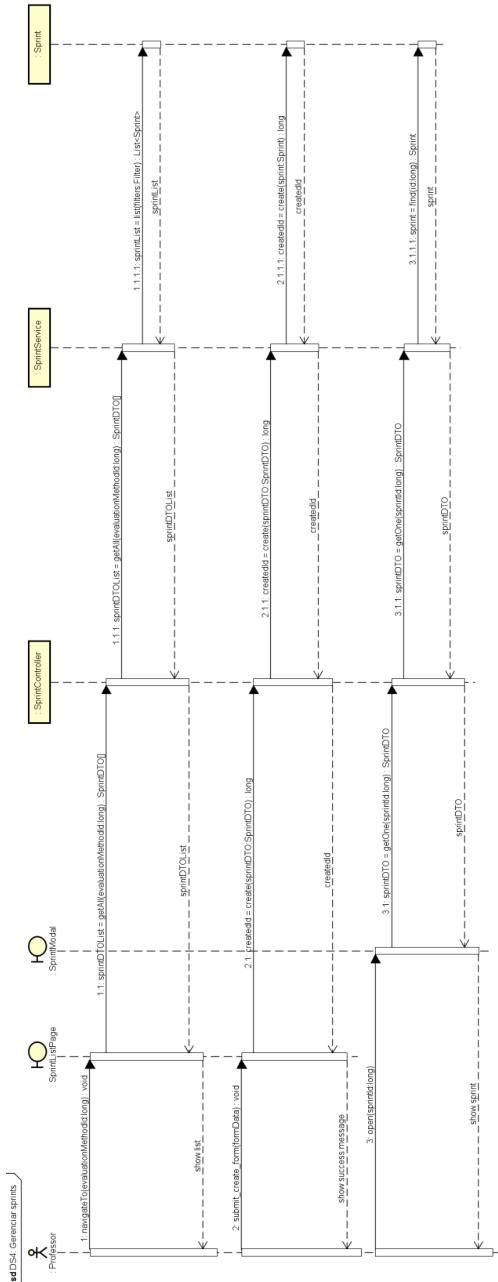
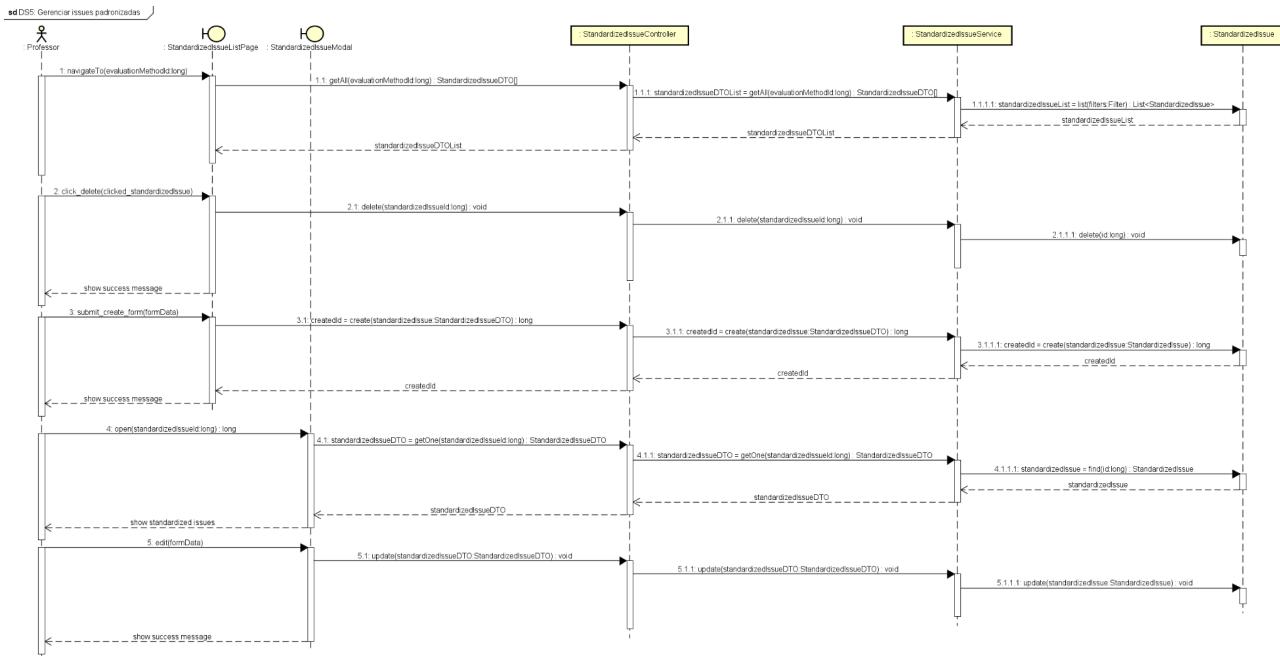


Figura 16. DS 4: Gerenciar sprints

A Figura 17 representa o diagrama de sequência responsável pelo fluxo de gerenciar *issues* padronizadas. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de criar, atualizar, deletar, buscar e listar *issues* padronizadas. A partir disso, as mensagens chegam ao componente de execução *StandardizedIssueController* que trata os dados da requisição e, com isso, enviar para o componente de execução *StandardizedIssueService* que manipula a entidade *StandardizedIssue* dentro do sistema conforme o objetivo requerido pelo usuário. Esse diagrama contempla os casos de uso UC10, UC39, UC40, UC41 e UC42.

Figura 17. DS 5: Gerenciar *issues* padronizadas

A Figura 18 representa o diagrama de sequência responsável pelo fluxo de gerenciar repositórios de trabalhos. Nesse fluxo, o usuário Professor envia mensagens síncronas ao sistema com o objetivo de sincronizar, buscar, listar, atualizar, calcular métricas e qualidade do código, buscar histórico de commits e consistência das entregas de um repositório de trabalho. A partir disso, as mensagens chegam ao componente de execução *RepositoryController* que trata os dados da requisição e, com isso, enviar para o componente de execução *RepositoryService* que manipula a entidade *Repository* dentro do sistema conforme o objetivo requerido pelo usuário. Esse diagrama contempla os caso de uso UC2, UC11, UC12, UC24, UC25, UC26, UC29, UC31 e UC32.

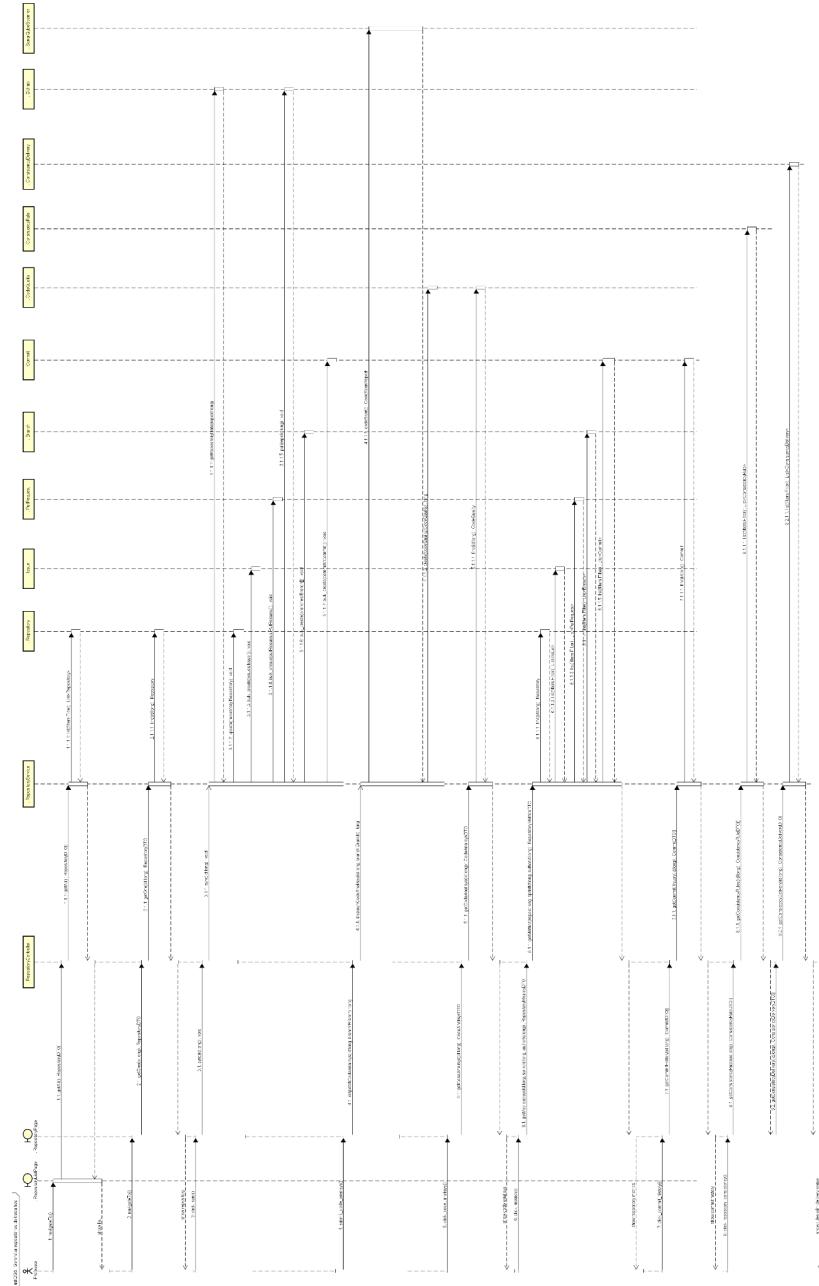


Figura 18. DS 6: Gerenciar repositórios de trabalhos

3.3 Diagramas de Comunicação

Nesta seção, são apresentados diagramas de comunicação que modelam as trocas de mensagens dos casos de uso do sistema implementado. Esses diagramas representam as interações entre componentes do sistema para realizar uma determinada funcionalidade. Dessa forma, os principais fluxos da aplicação são representados como meio de documentar as comunicações que os compõem.

A Figura 19 representa o diagrama de comunicação responsável pelo fluxo de autenticar na plataforma. Nesse fluxo, o usuário Professor inicia a comunicação com o sistema pela *AuthPage*.

com o objetivo de autenticar-se na plataforma por meio de uma autorização. Diante disso, a comunicação é recebida pelo *AuthController* que irá se comunicar com sistema externo GitHub com intuito de validar as credenciais informadas. Ao sucesso da validação, a autorização é por uma solicitação feita ao *AuthService*, assim, permitindo o usuário *User* continuar a utilização da plataforma. Esse diagrama contempla o caso de uso UC1.

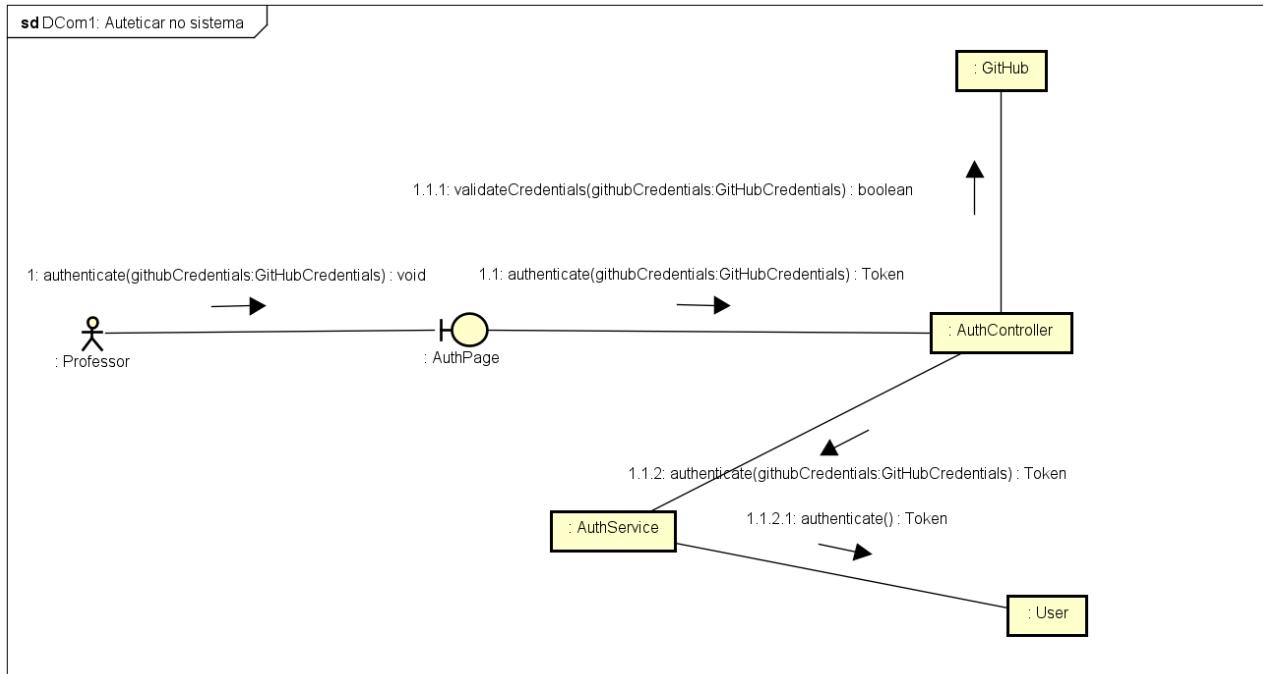


Figura 19. DCom 1: Autenticar na plataforma

A Figura 20 representa o diagrama de comunicação responsável pelo fluxo de gerenciar métodos avaliativos. Nesse fluxo, o usuário Professor inicia a comunicação com o sistema pela *EvaluationMethodListPage* ou *EvaluationMethodPage* com o objetivo de criar, atualizar, buscar, listar, desativar, duplicar ou adicionar um repositório a um método avaliativo. Diante disso, a comunicação é recebida pelo *EvaluationMethodController* que irá validar os dados e detectar qual ação que deve ser realizada. Ao definir qual ação, a comunicação com o *EvaluationMethodService* é iniciada, assim, executando a ação requerida pelo Professor. Esse diagrama contempla os casos de uso UC3, UC4, UC5, UC6, UC21, UC33, UC34 e UC35.

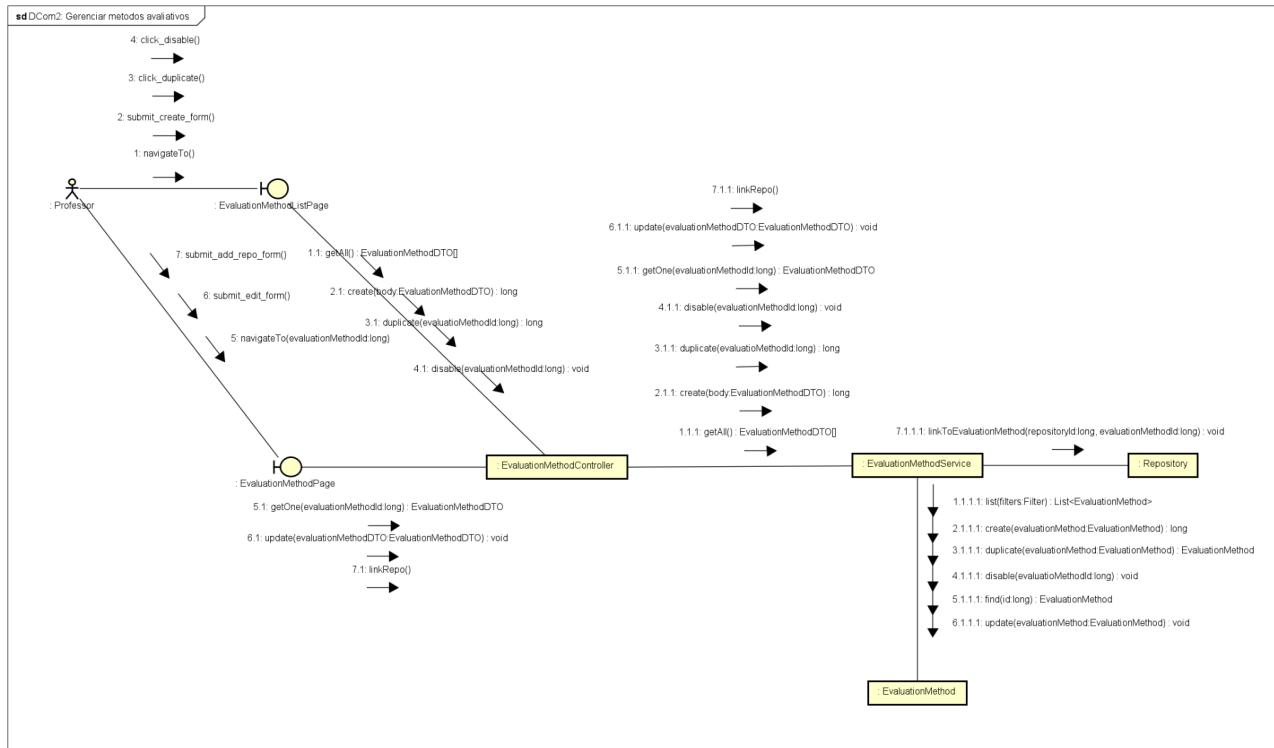


Figura 20. DCom 2: Gerenciar métodos avaliativos

A Figura 21 representa o diagrama de comunicação responsável pelo fluxo de gerenciar regras de consistência. Nesse fluxo, o usuário Professor inicia a comunicação com o sistema pela *ConsistencyRuleListPage* ou *ConsistencyRuleModal* com o objetivo de criar, atualizar, buscar e listar regras de consistência. Diante disso, a comunicação é recebida pelo *ConsistencyRuleController* que irá validar os dados e detectar qual ação que deve ser realizada. Ao definir qual a ação, a comunicação com o *ConsistencyRuleService* é iniciada, assim, executando a ação requerida pelo Professor. Esse diagrama contempla os casos de uso UC6, UC7, UC8, UC9, UC22 e UC36.

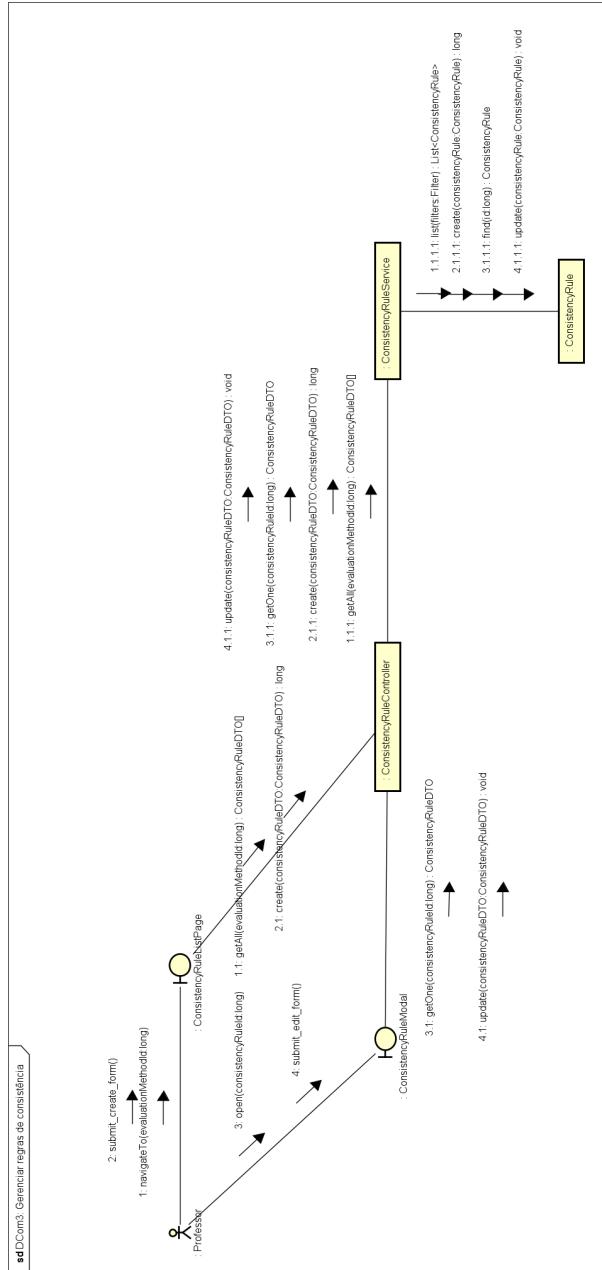
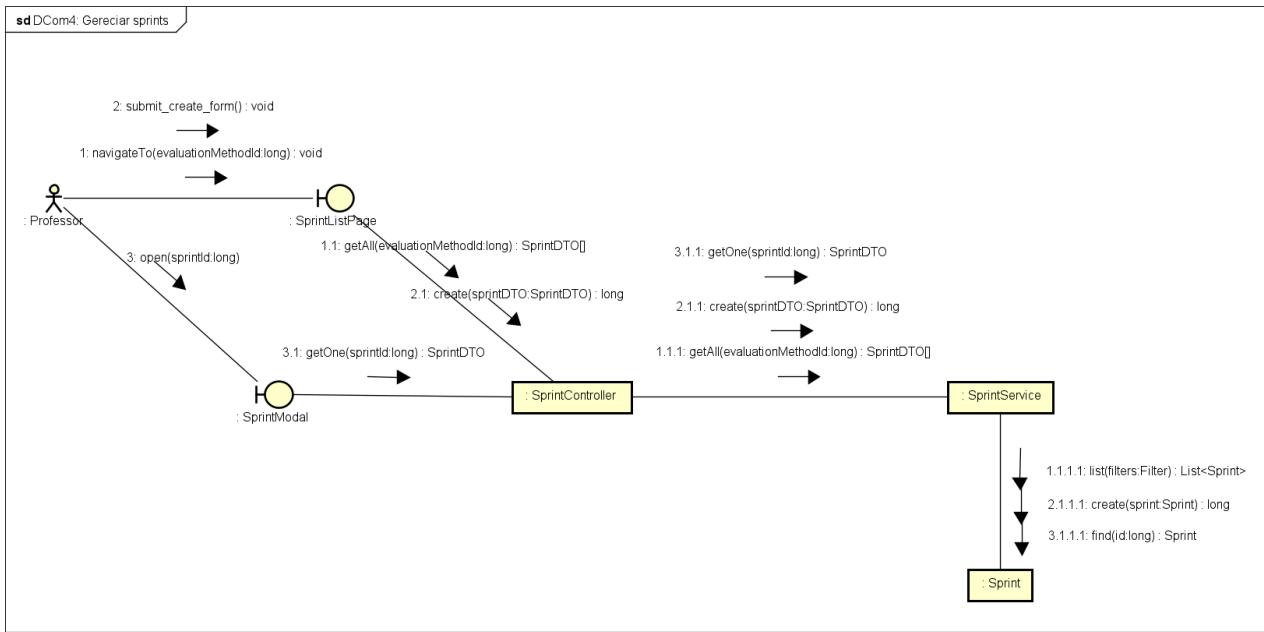
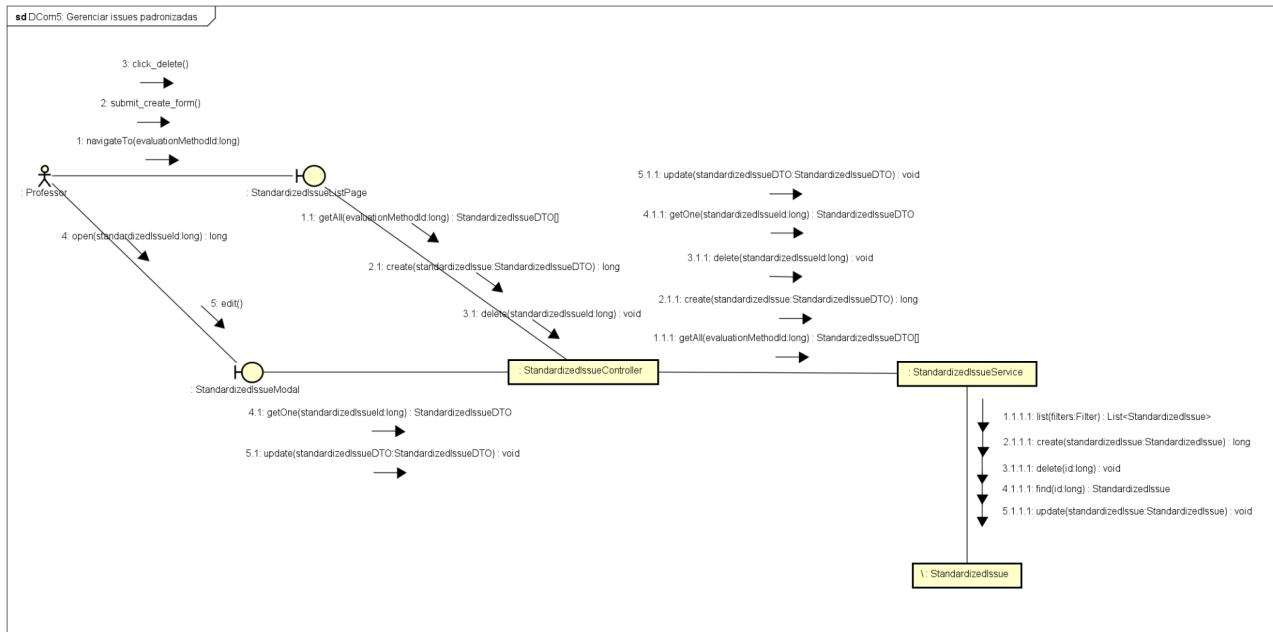


Figura 21. DCom 3: Gerenciar regras de consistência

A Figura 22 representa o diagrama de comunicação responsável pelo fluxo de gerenciar *sprints*. Nesse fluxo, o usuário Professor inicia a comunicação com o sistema pela *SprintListPage* ou *SprintModal* com o objetivo de criar, buscar e listar *sprints*. Diante disso, a comunicação é recebida pelo *SprintController* que irá validar os dados e detectar qual ação que deve ser realizada. Ao definir qual a ação, a comunicação com o *SprintService* é iniciada, assim, executando a ação requerida pelo Professor. Esse diagrama contempla os casos de uso UC21, UC22, UC37 e UC38.

Figura 22. DCom 4: Gerenciar *sprints*

A Figura 23 representa o diagrama de comunicação responsável pelo fluxo de gerenciar *issues* padronizadas. Nesse fluxo, o usuário Professor inicia a comunicação com o sistema pela *StandardizedIssueListPage* ou *StandardizedIssueModal* com o objetivo de criar, atualizar, deletar, buscar e listar *issues* padronizadas. Diante disso, a comunicação é recebida pelo *StandardizedIssueController* que irá validar os dados e detectar qual ação que deve ser realizada. Ao definir qual a ação, a comunicação com o *StandardizedIssueService* é iniciada, assim, executando a ação requerida pelo Professor. Esse diagrama contempla os casos de uso UC10, UC39, UC40, UC41 e UC42.

Figura 23. DCom 5: Gerenciar *issues* padronizadas

A Figura 24 representa o diagrama de comunicação responsável pelo fluxo de gerenciar repositórios de trabalhos. Nesse fluxo, o usuário Professor inicia a comunicação com o sistema pela *RepositoryListPage* ou *RepositoryPage* com o objetivo de criar, atualizar, buscar e listar regras de consistência. Diante disso, a comunicação é recebida pelo *RepositoryController* que irá validar os dados e detectar qual ação que deve ser realizada. Ao definir qual a ação, a comunicação com o *RepositoryService* é iniciada, assim, executando a ação requerida pelo Professor. Esse diagrama contempla os casos de uso UC2, UC11, UC12, UC24, UC25, UC26, UC29, UC31 e UC32.

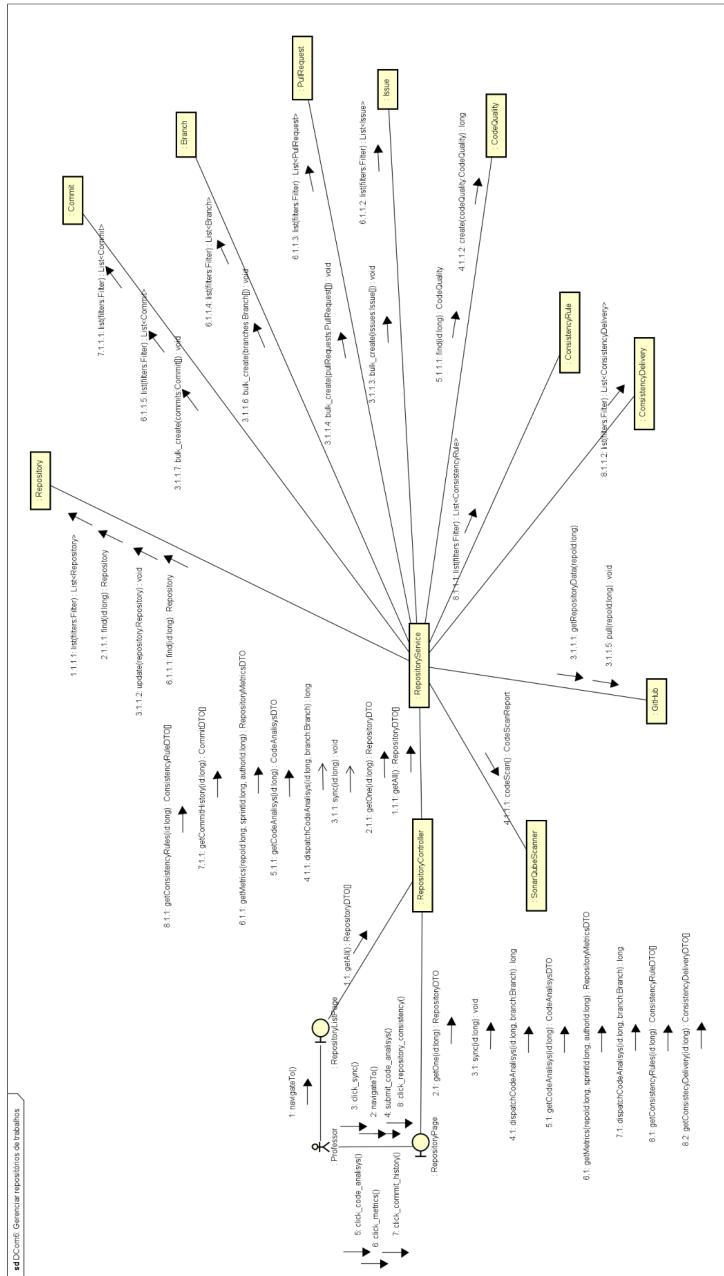


Figura 24. DCom 6: Gerenciar repositórios de trabalhos

3.4 Arquitetura

Nesta seção, é apresentada a modelagem da arquitetura lógica do sistema. Essa modelagem foi feita por meio do diagrama de pacotes, o qual descreve a estrutura organizacional do sistema, mostrando como os elementos do sistema estão organizados e hierarquizados em pacotes e como esses pacotes se relacionam entre si.

A Figura 25 representa o diagrama de pacotes da plataforma, que possui o pacote DockerCompose o qual contém os quatro principais pacotes do sistema, sendo eles o

SupportPlatform-Service, View, SonarQube e JobScheduler-Service. O DockerCompose refere-se ao pacote da ferramenta de administração de contêineres internos da plataforma. Já o pacote SupportPlatform-Service é o responsável pelas funções de manipulação dos dados dentro da plataforma, esse pacote depende do pacote SonarQube para efetuar as operações de análise estática de código. O pacote View possui as interfaces que os usuários finais utilizam, sendo dependente direto do SupportPlatform-Service para manipular os dados. Por fim, o pacote JobScheduler-Service é responsável pela sincronização da base de dados com as informações provenientes do GitHub, também sendo dependente direto do SupportPlatform-Service para manipular os dados.

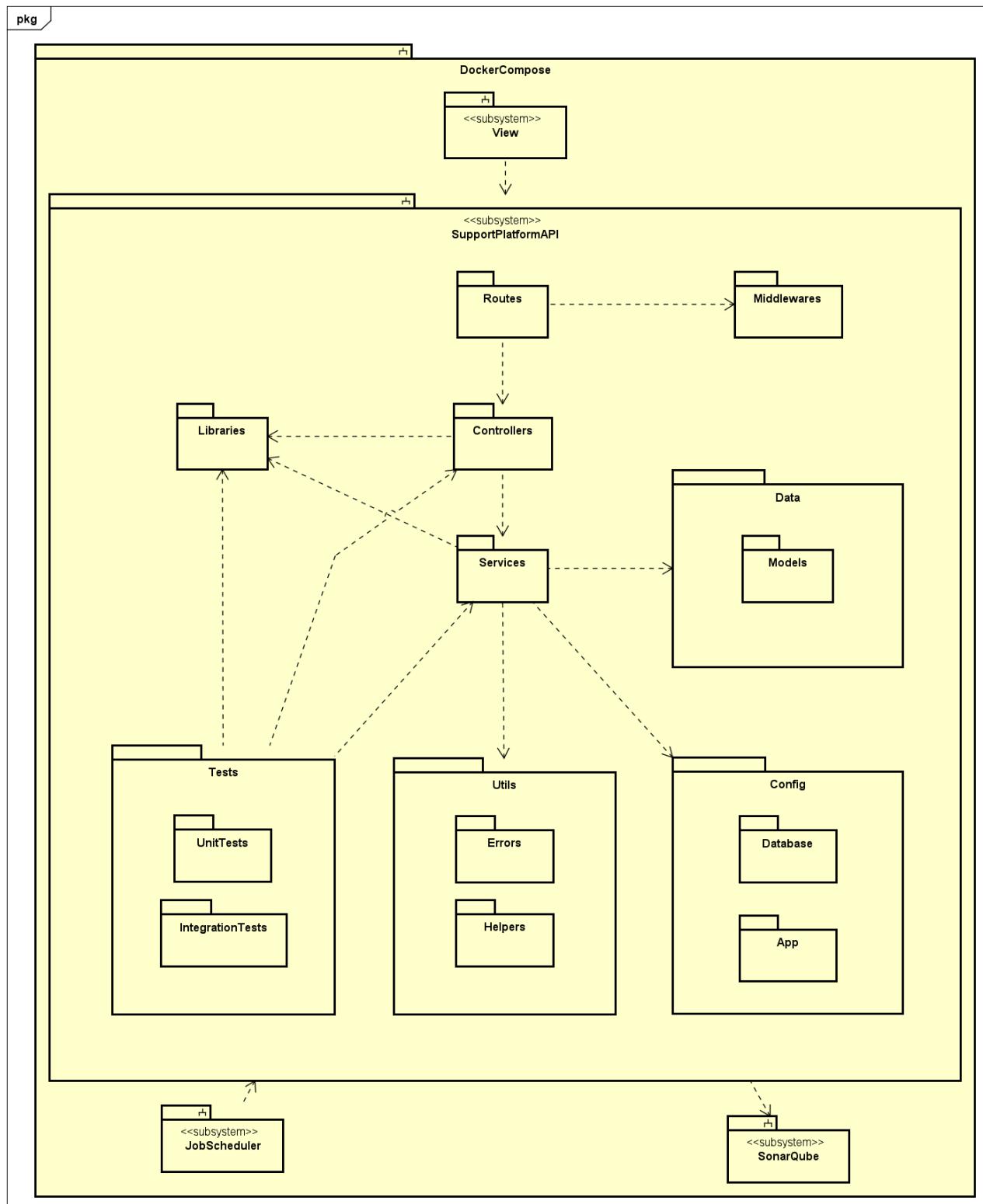


Figura 25. Diagrama de Pacotes do sistema

3.5 Diagramas de Estados

Nesta seção, são apresentados os diagramas de estados do sistema implementado. Esses diagramas modelam os possíveis estados que um objeto de uma determinada entidade pode estar, e especificam as transições entre eles. A Figura 26 apresenta o diagrama de estados dos objetos da entidade *Code Quality*. Nesse diagrama, é apresentado que o objeto se inicia com o *status* “analisando”. Assim que a análise de qualidade de código é finalizada, seu *status* se altera para “analisado”. Caso ocorra algum erro durante a análise, o *status* passa a ser “falhou”. Nos dois últimos casos, esse é o estado final que o objeto pode assumir, não havendo nenhuma transição possível uma vez que ele chega em um desses estados. Esse diagrama se relaciona com o caso de uso UC11.

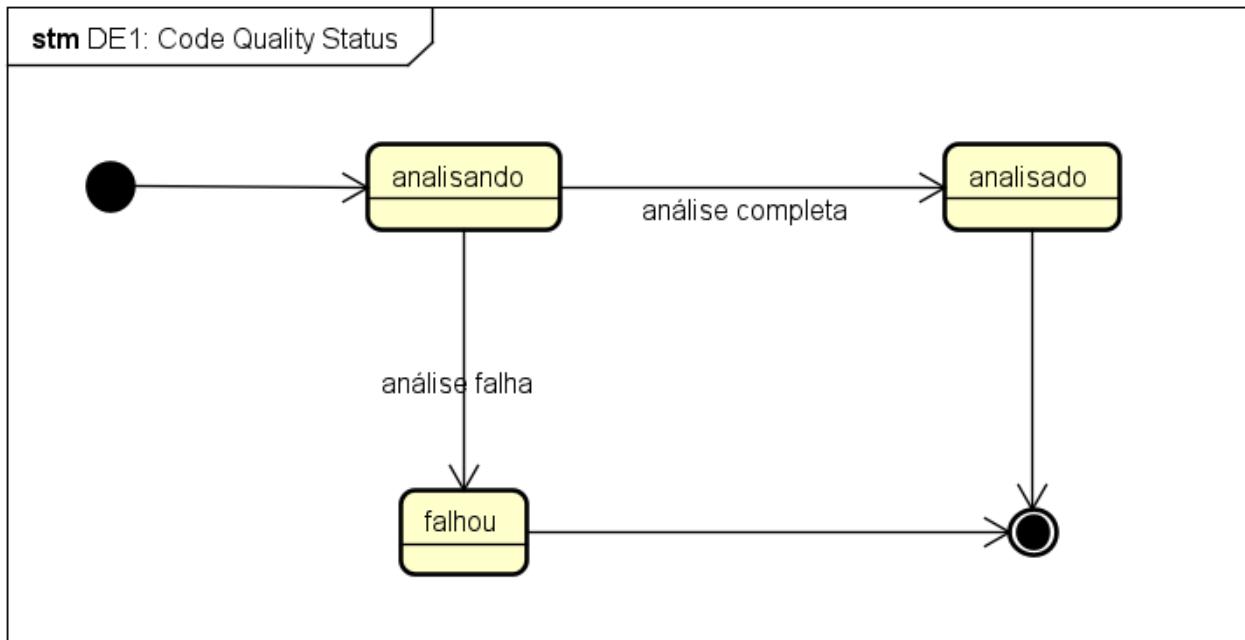
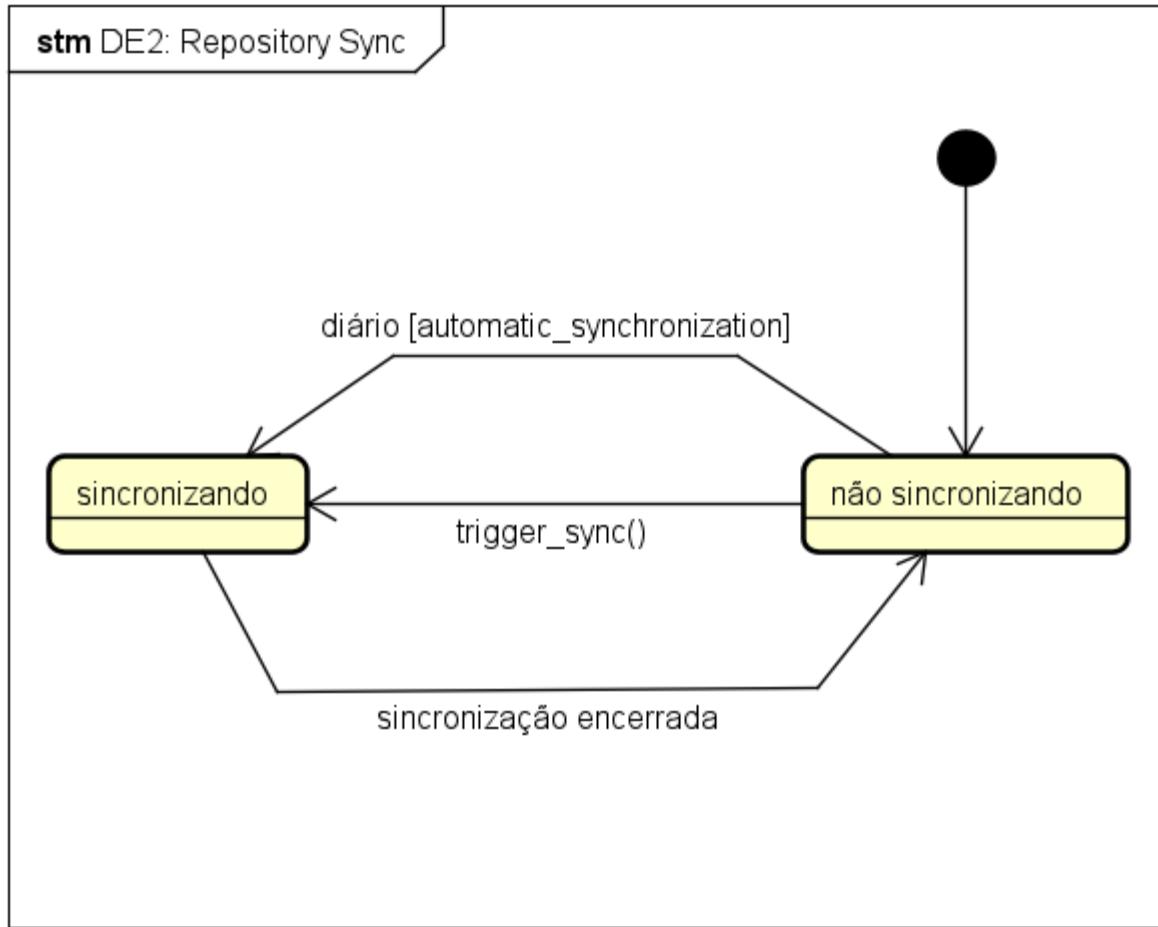


Figura 26. Diagrama de Estados da Entidade *Code Quality*

A Figura 27 apresenta o diagrama de estados dos objetos da entidade *Repository*. Nesse diagrama, é apresentado que o objeto inicia com a propriedade *sync* como “não sincronizando”. Há dois eventos que provocam a transição para o estado “sincronizando”. O primeiro é o disparo manual da função “*trigger_sync*”, e o segundo ocorre uma vez por dia, caso o atributo “*automatic_synchronization*” seja verdadeiro. Uma vez que se encontra no estado “sincronizando”, ele retorna ao estado “não sincronizando” assim que a sincronização é encerrada. Esse diagrama se relaciona com o caso de uso UC 31.

Figura 27. Diagrama de Estados da Entidade *Repository*

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 28 é apresentado o diagrama de componentes da plataforma. O principal componente apresentado é o Web Server, que se relaciona com todos os outros componentes. Ele é o componente que provê a API para a aplicação *web*, comunica-se com o banco de dados, com a API externa do GitHub, além de requisitar à instância do SonarQube e ser chamado pelo JobScheduler-Service, que agenda rotinas de execução.

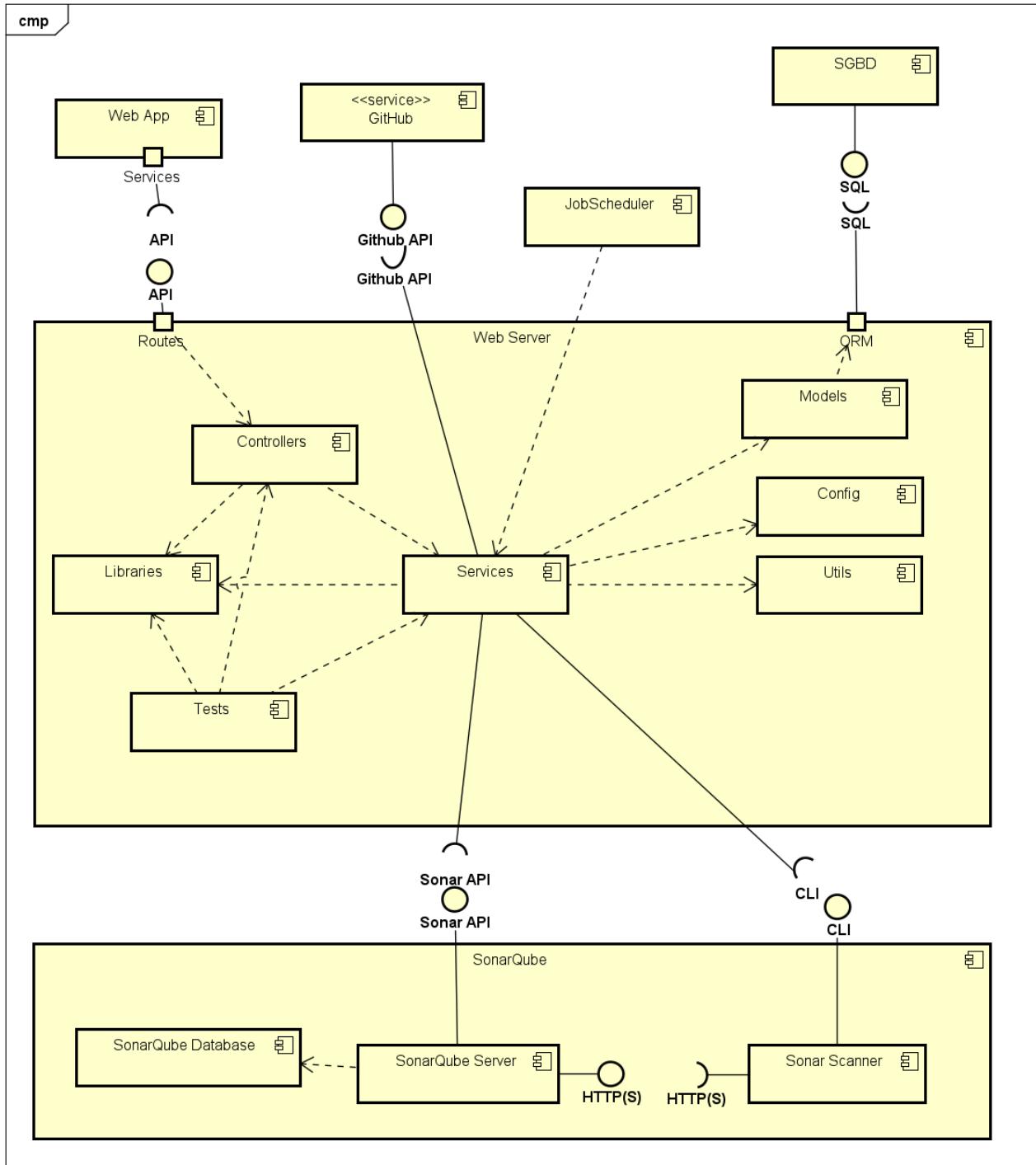


Figura 28. Diagrama de Componentes

Na Figura 29 é apresentado o diagrama de implantação da plataforma. Nele, são apresentados os nós de processamento para implantação e uso da mesma. A aplicação web, executada usando um *web browser* na máquina cliente, irá baixar os arquivos estáticos do Web Server e fazer requisições à Application Server, através do protocolo HTTPS. Os dois servidores estão em máquinas isoladas um do outro. Outras instâncias necessárias são o SonarQube e o banco

de dados. Essas também são isoladas, e ambas são requisitadas pelo Application Server através do protocolo TCP/IP.

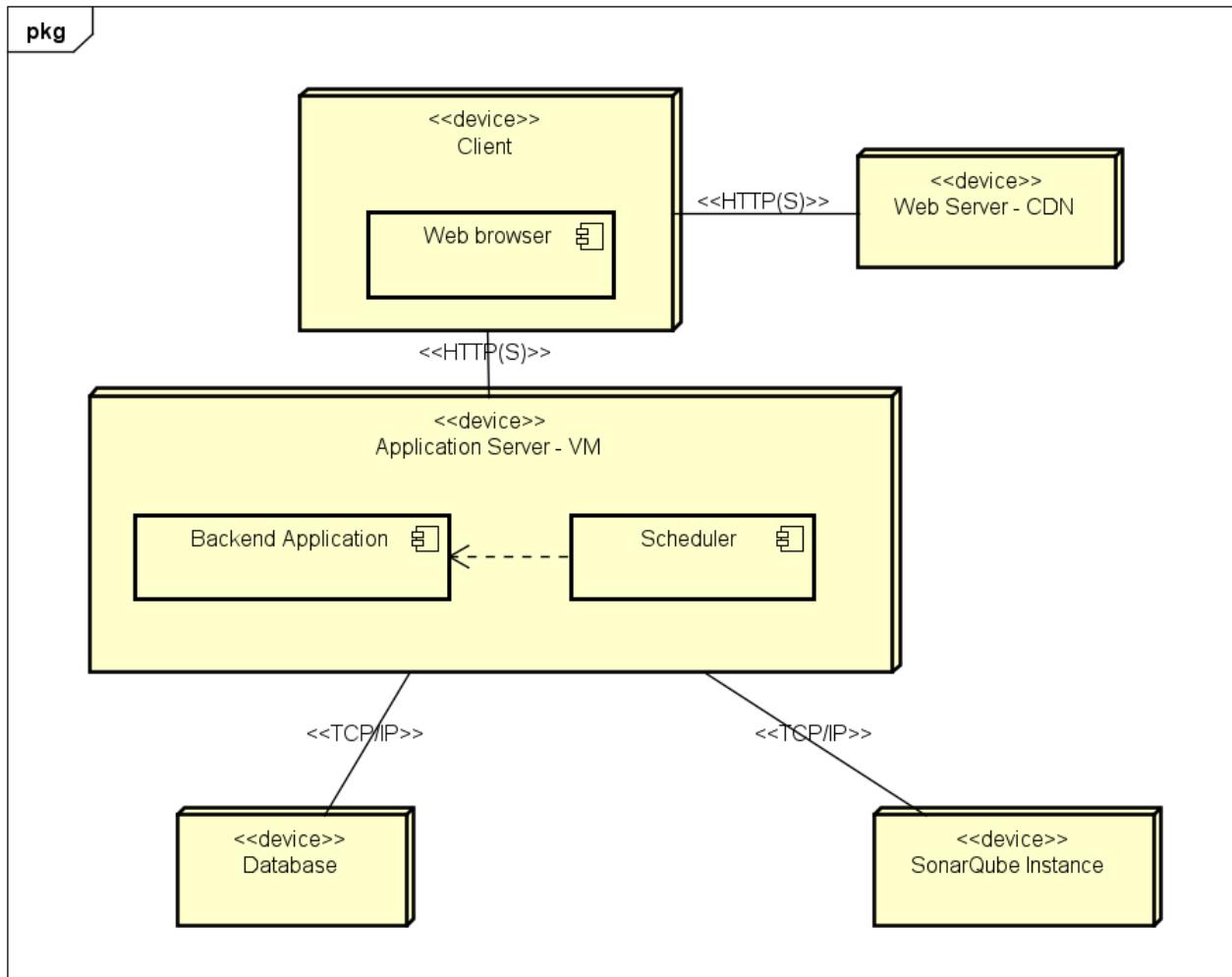


Figura 29. Diagrama de Implantação

4. Projeto de Interface com Usuário

Esta seção visa apresentar e descrever as principais interfaces de usuário da plataforma desenvolvida neste trabalho. Para isso, foram desenvolvidos *mockups* de alta fidelidade por meio da ferramenta Figma³ e as bibliotecas de *design* Primer⁴ e ChartJs⁵. Essas bibliotecas foram escolhidas, por permitirem a criação de elementos gráficos semelhantes ao GitHub. Diante disso, as interfaces

³ <https://www.figma.com/>

⁴ <https://primer.style/>

⁵ <https://www.chartjs.org/>

foram projetadas relacionando-as com os casos de uso definidos na Seção 2.3.1 com objetivo de mapear todas as funcionalidades necessárias para cumprimento dos requisitos especificados.

A Figura 30 representa a tela de *login* para usuário não autenticadas na aplicação. Essa tela redireciona o professor para a tela da Figura 31 após uma autenticação com sucesso utilizando as credenciais do GitHub. Essa interface contempla o caso de uso UC1.

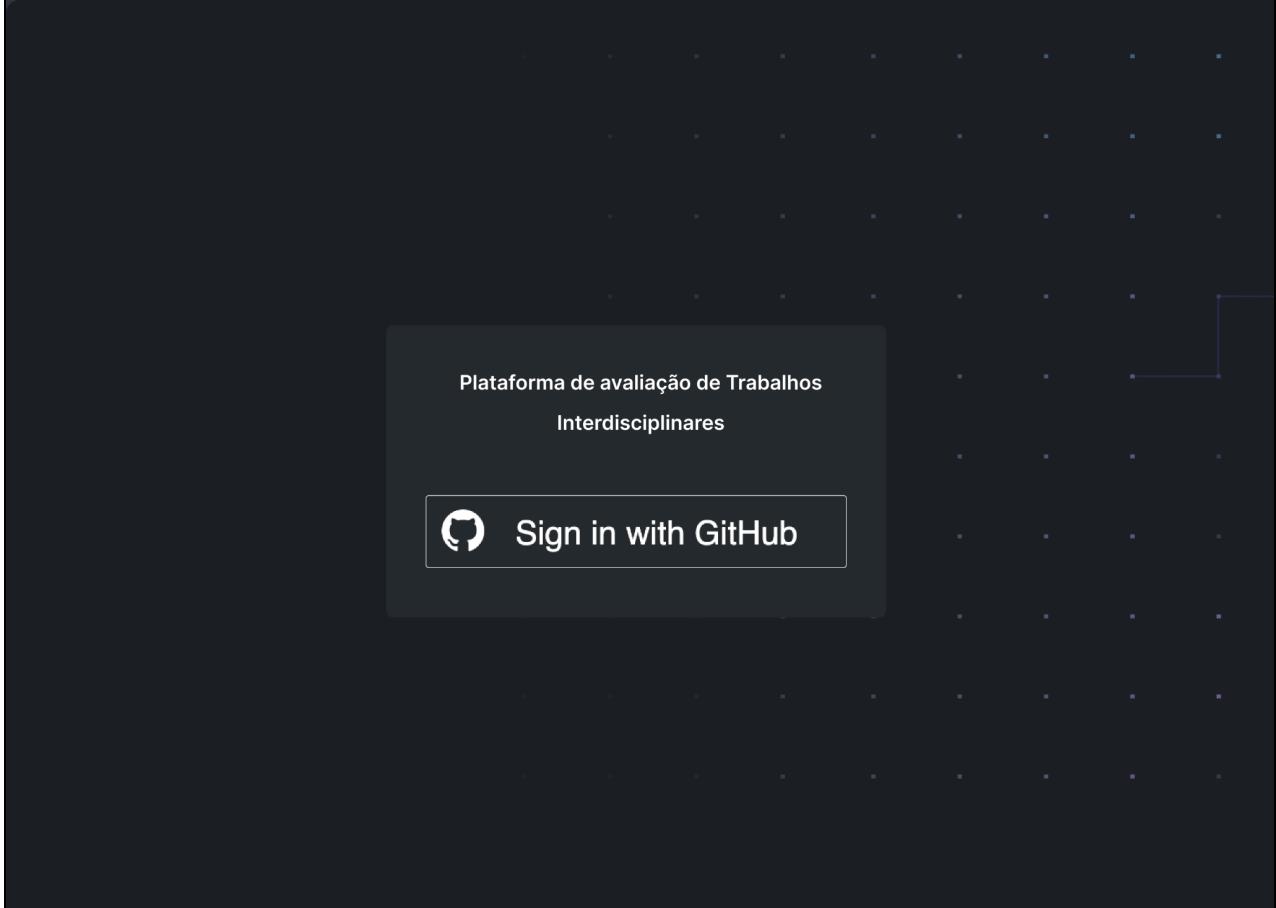


Figura 30. Tela de *login* na plataforma

A Figura 31 é a página inicial da plataforma que será aberta para os professores logo após a autenticação do usuário. Essa e todas as páginas do sistema possuem um *header* padrão com o ícone da plataforma, uma barra de pesquisa para buscar repositório de um trabalho pelo nome e os botões para visualizar a lista de repositórios de trabalhos e lista de métodos avaliativos, também possui um *footer* padrão com o *link* do repositório do GitHub deste TCC. Além disso, nessa página há a listagem de métodos avaliativos cadastrados na plataforma, sendo possível filtrar pelo nome do método avaliativo, abri-los e duplicá-los para facilitar na reutilização das configurações em vários semestres letivos. Essa interface contempla os casos de uso UC3, UC33 e UC35.

Buscar trabalho... Repositórios de trabalhos Métodos Avaliativos

Filtrar Q

Duplicar Editar

< Anterior 1 2 3 4 Próxima >

© 2023 Plataforma de Apoio às Avaliações de Projetos GitHub. Repertório

Figura 31. Tela de listagem de métodos avaliativos

A Figura 32 representa a página de gerenciamento dos repositórios de um método avaliativo. Nessa página é possível adicionar e filtrar quais repositórios estão sendo avaliados por esse método e duplicar o método avaliativo. Além disso, é possível navegar na tela para visualizar as regras de consistência, *sprints* e *issues* padronizadas do método avaliativo que está sendo gerenciado. Essa interface contempla os caso de usos UC5, UC34 e UC35.

The screenshot shows the 'Repositórios' (Repositories) section of the 'Método Avaliativo: Trabalho Interdisciplinar1 - 01/2023' page. At the top, there are navigation links for 'Repositórios de trabalhos' and 'Métodos Avaliativos'. A search bar labeled 'Buscar trabalho...' is present. On the right, there are buttons for 'Duplicar' (Duplicate) and 'Adicionar repositório' (Add repository). The main area displays a list of repositories, each with a preview icon, the name 'nome-do-rep', and two buttons: 'Sincronizar' (Sync) and 'Abrir' (Open). A 'Filtrar' (Filter) button and a search input field are located above the repository list. Below the list, there is a pagination control with buttons for 'Anterior' (Previous), page numbers 1, 2, 3, 4, and 'Próxima' (Next). At the bottom, there is a footer with the GitHub logo and the text '© 2023 Plataforma de Apoio às Avaliações de Projetos GitHub.' and a 'Repositório' link.

Figura 32. Tela dos repositórios de um método avaliativo

A Figura 33 representa a página de gerenciamento de um método avaliativo, mais especificamente o gerenciamento de suas regras de consistência. Nessa página é possível ver, editar e cadastrar as regras de consistência do método avaliativo. Essa interface contempla os casos de uso UC6, UC7, UC8, UC9 e UC36.

The screenshot shows a web-based application interface for managing project evaluations. At the top, there's a navigation bar with icons for user profile, search, and repository management. Below the navigation, the title "Método Avaliativo: Trabalho Interdisciplinar! - 01/2023" is displayed. On the left, a sidebar lists categories: "Repositórios", "Regras de consistência" (which is selected and highlighted in grey), "Sprints", and "Issues padronizadas". A search bar with filters is present. The main content area displays a list of consistency rules, each with a file icon, name, sprint status, and an "Editar" button. The rules listed are: citation.cff, Apresentacao/video.mp4, Documentacao/arquivo_de_arquitetura.md, diretório/arquivo.extensão, diretório/arquivo.extensão, diretório/arquivo.extensão, diretório/arquivo.extensão, diretório/arquivo.extensão, and diretório/arquivo.extensão. Below the list is a pagination bar with buttons for 'Anterior', page numbers 1, 2, 3, 4, and 'Próxima'.

Figura 33. Tela das regras de consistência de um método avaliativo

A Figura 34 apresenta o modal de cadastro de uma nova regra de consistência em um método avaliativo na plataforma. Esse modal será aberto ao clicar no botão “Criar regra de consistência” presente na Figura 33. Nesse modal é necessário inserir o diretório e nome do arquivo que será validado, a *sprint* de entrega desse arquivo, a *issue* padronizada caso a regra de consistência tenha sido quebrada e quais as possíveis extensões de arquivos aceitas por essa regra, sendo válido observar que caso tenha a extensão “cff” aparece o alerta que será validado a estrutura de arquivo de citação. Além disso, o modal dessa tela possui dois botões para melhorar a usabilidade, sendo eles para criar *sprints* e *issues* padronizadas sem ser necessário trocar de tela. Essa interface contempla os casos de uso UC9, UC10, UC22 e UC29.

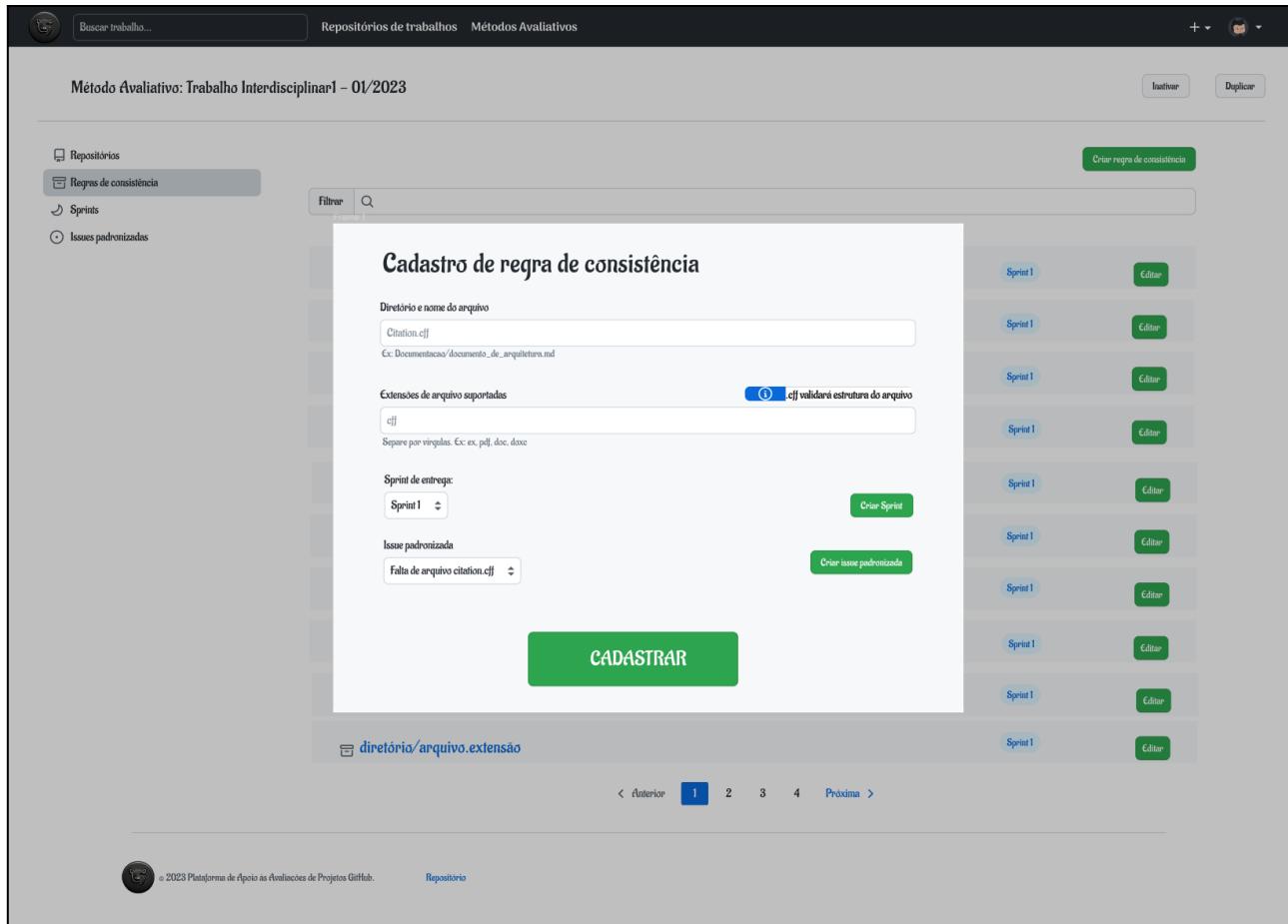


Figura 34. Tela de modal de cadastro de regra de consistência

A Figura 35 representa a página de gerenciamento das *sprints* de um método avaliativo. Nessa página é possível criar, editar e filtrar as *sprints* do método avaliativo. Essa interface contempla o caso de uso UC21.

The screenshot shows a web application interface for managing sprints in a project evaluation method. At the top, there's a header with a logo, a search bar labeled 'Buscar trabalho...', and navigation links for 'Repositórios de trabalhos' and 'Métodos Avaliativos'. On the right side of the header are buttons for 'Iniciar' and 'Duplicar'. Below the header, the title 'Método Avaliativo: Trabalho Interdisciplinar1 - 01/2023' is displayed. On the left, a sidebar menu includes 'Repositórios', 'Regras de consistência', 'Sprints' (which is selected and highlighted in grey), and 'Issues padronizadas'. A search bar with 'Filtrar' and a magnifying glass icon is positioned next to the sidebar. To the right of the sidebar, a green button labeled 'Criar Sprint' is visible. The main content area lists ten sprints, each with a blue circular icon followed by the sprint name and an 'Editar' button: Sprint 1, Sprint 2, Sprint 3, Sprint 4, Sprint 5, Sprint 6, Sprint 7, Sprint 8, Sprint 9, and Sprint 10. At the bottom of the list is a navigation bar with buttons for '< Anterior', '1' (selected), '2', '3', '4', and 'Próxima >'.

Figura 35. Tela de *sprints* de um método avaliativo

A Figura 36 representa a página de gerenciamento das *issues* padronizadas de um método avaliativo. Nessa página é possível criar, editar e filtrar as *issues* padronizadas do método avaliativo. Essa interface contempla os casos de uso UC10, UC23, UC39, UC40, UC41 e UC42.

The screenshot shows a web application interface for managing project evaluations. At the top, there's a navigation bar with icons for user profile, search, and repository management. Below the navigation, the title "Método Avaliativo: Trabalho Interdisciplinar1 – 01/2023" is displayed. On the left, a sidebar lists categories: "Repositórios", "Regras de consistência", "Sprints", and "Issues padronizadas", with "Issues padronizadas" currently selected. A search bar with filters is present. The main area displays a table of standardized issues, each with a status "Sprint 1" and an "Editar" button. The issues listed are all variations of "nome da issue padronizada". At the bottom, there are navigation links for "Anterior" (page 1), "Próxima" (page 2).

Figura 36. Tela de *issues* padronizadas de um método avaliativo

A Figura 37 representa a página onde são listados os repositórios pertencentes à organização ICEI-PUC-Minas-PPLES-TI do GitHub, possibilitando os professores visualizarem os nomes dos repositórios e dos seus respectivos métodos avaliativos. Por fim, também há os botões de Sincronizar para atualizar a base de dados de cada repositório com do GitHub e o Abrir que possibilita abrir a visão geral específica do repositório de um trabalho. Essa interface contempla os casos de uso UC2, UC27 e UC31.

Figura 37. Tela de listagem de repositórios de trabalhos

A Figura 38 representa a tela de visão geral das informações de um trabalho. Nessa tela os professores podem visualizar as métricas por tipo de informação, *branches*, *sprints* de cada contribuidor do repositório ou do repositório inteiro. Além disso, poderá ativar e desativar a sincronização automática, sincronizar manualmente, redirecionar para o repositório, visualizar contribuidores e método avaliativo. Ademais, nessa tela é possível mudar a visualização entre métricas do repositório, qualidade do código, históricos de *commits*, conformidade com regras de consistência e configurações do trabalho. Quando acionado o menu de métricas, são apresentados diversos indicadores e gráficos a respeito da contribuição do grupo e dos alunos nas entregas, com a possibilidade de filtrar por *sprint* e pela *branch* específica. Essa interface contempla os casos de uso UC26, UC27, UC12, UC13, UC14, UC15, UC16, UC17, UC18, UC19 e UC20.

No contexto apresentado pela Figura 38, é crucial mencionar a coluna e filtro dedicado a representar *commits* sem contribuidor nos gráficos de métricas. Esta coluna e filtro foi estabelecida para abordar situações em que *commits* são feitos, mas, por diversas razões, como endereços de email não vinculados a uma conta no GitHub, *commits* originados de bifurcações antes de serem integrados ou *commits* que não atendem certos critérios estabelecidos pelo GitHub⁶. Com essa

⁶

<https://docs.github.com/pt/account-and-profile/setting-up-and-managing-your-github-profile/managing-contribution-settings-on-your-profile/why-are-my-contributions-not-showing-up-on-my-profile>

coluna, os professores têm uma visão mais abrangente e transparente das atividades do repositório, garantindo que todo o trabalho seja reconhecido e considerado, mesmo aquele não diretamente vinculado a um perfil específico. A presença desta coluna enfatiza a importância de reconhecer todas as contribuições, incluindo aquelas que, por razões técnicas ou administrativas, não estão diretamente associadas a uma identidade particular.

Com relação à métrica de qualidade da descrição de *commit*, ela é calculada com base no número de caracteres da mensagem⁷. Descrições com menos de 10 caracteres são classificadas como “Péssima”, indicando uma explicação insuficiente. Mensagens com menos de 20 caracteres são rotuladas como “Ruim”, enquanto aquelas com menos de 40 caracteres são consideradas “Regular”. Uma descrição de *commit* que possui até 100 caracteres é vista como “Ideal”, pois geralmente proporciona uma clara e concisa explicação da alteração. No entanto, mensagens que excedem 100 caracteres são categorizadas como “Excessiva”, sinalizando que a descrição pode ser muito detalhada ou complexa. Essa métrica visa promover boas práticas entre os contribuidores, incentivando descrições claras, concisas e informativas para cada *commit*. As descrições de *commit* que iniciam com “Merge branch”, “Merge remote-tracking branch” e “Merge pull request” são ignoradas, por serem descrições comumente geradas automaticamente pelo GitHub.

Uma particularidade da métrica de arquivos contribuídos nesta plataforma que pode confundir é que o número de arquivos contribuídos seja superior ao total de arquivos presentes no GitHub. Isso ocorre devido à metodologia de sincronização diária adotada pelo sistema. Cada dia, ao sincronizar os arquivos, o sistema não apenas identifica novas adições, mas também reconhece mudanças como renomeações, realocações e modificações repetidas no mesmo arquivo em diversos *commits* como contribuições distintas. Por exemplo, se um diretório é renomeado no GitHub, todos os arquivos contidos nele são interpretados pela plataforma como criações, mesmo que apenas o nome do diretório tenha mudado. Da mesma forma, se um arquivo é modificado várias vezes em *commits* separados, cada modificação é contabilizada individualmente. Esta abordagem foi adotada para o sistema poder apresentar o desempenho e a atividade dos alunos ao longo do tempo, refletindo a intensidade e a natureza dinâmica de suas contribuições, em vez de apenas o estado contemporâneo de um repositório.

⁷ <https://www.conventionalcommits.org/en/v1.0.0/>

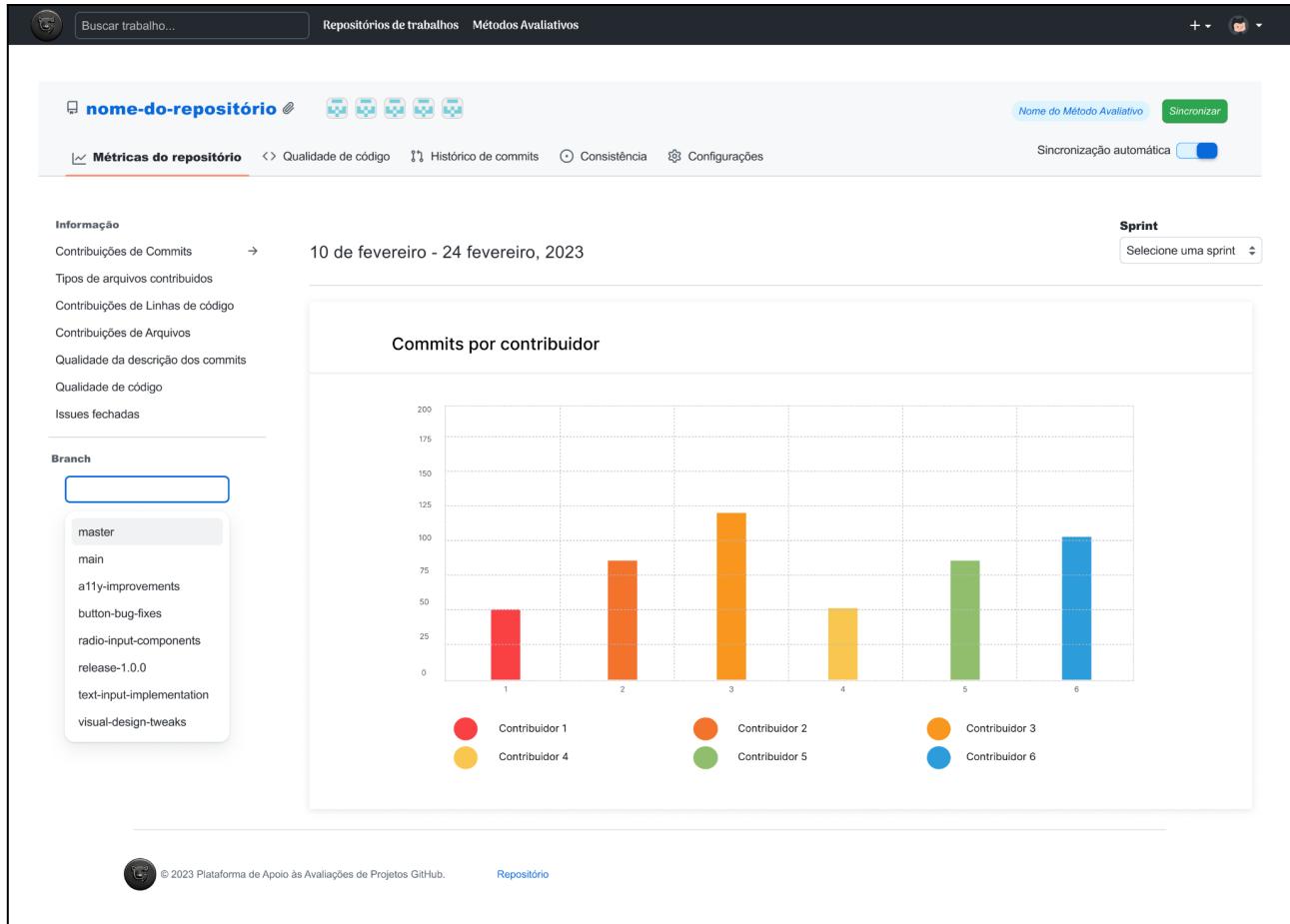


Figura 38. Tela de métricas de um trabalho

A Figura 39 apresenta a tela de visualização do trabalho, com a visualização de qualidade de código ativa. Nessa tela, é possível acionar uma análise estática de código, e, quando completa, são exibidos os resultados da mesma, provindos do SonarQube. Essa interface contempla o caso de uso UC11.

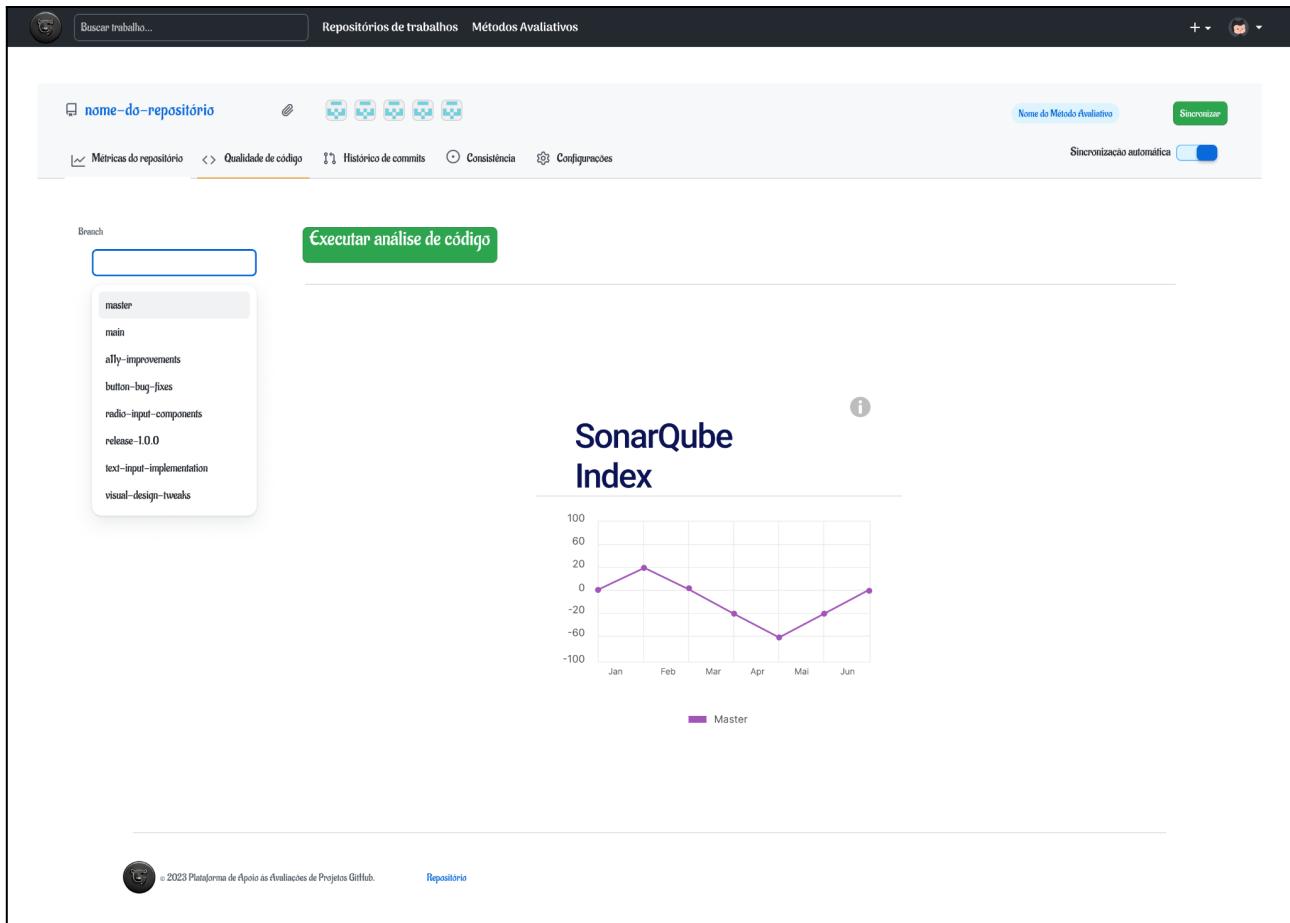


Figura 39. Tela de qualidade de código de um trabalho

A Figura 40 apresenta a tela de visualização do trabalho, com a visualização de histórico de *commits* ativa. Nessa tela, é possível visualizar uma lista de *commits* realizados no repositório, indicando quem o realizou e quando o fez, além de ser possível redirecionar para o *commit* no próprio *site* do GitHub. Essa interface contempla os casos de uso UC23 e UC24.

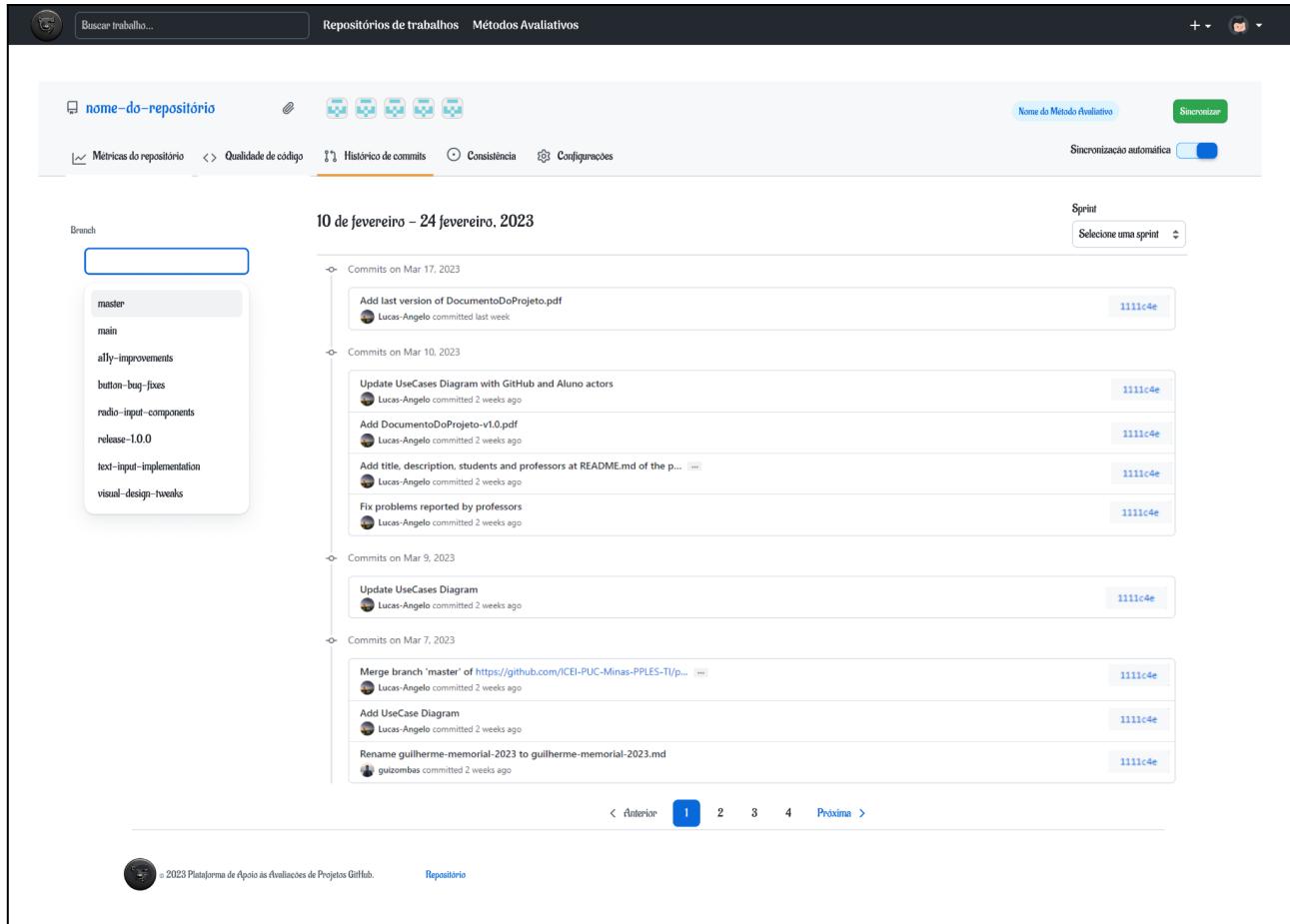


Figura 40. Tela do histórico de *commits* de um trabalho

A Figura 41 apresenta a tela de visualização do trabalho, com a visualização de consistência ativa. Nessa tela, é possível visualizar as regras de consistência configuradas para o trabalho a partir do método avaliativo vinculado. Cada regra apresenta uma cor indicando se ela foi entregue ou não, e se foi entregue com atraso. Além disso, é possível ainda filtrar a visualização das regras por *sprint* de entrega. Essa interface contempla os casos de uso UC25, UC28, UC29 e UC23.

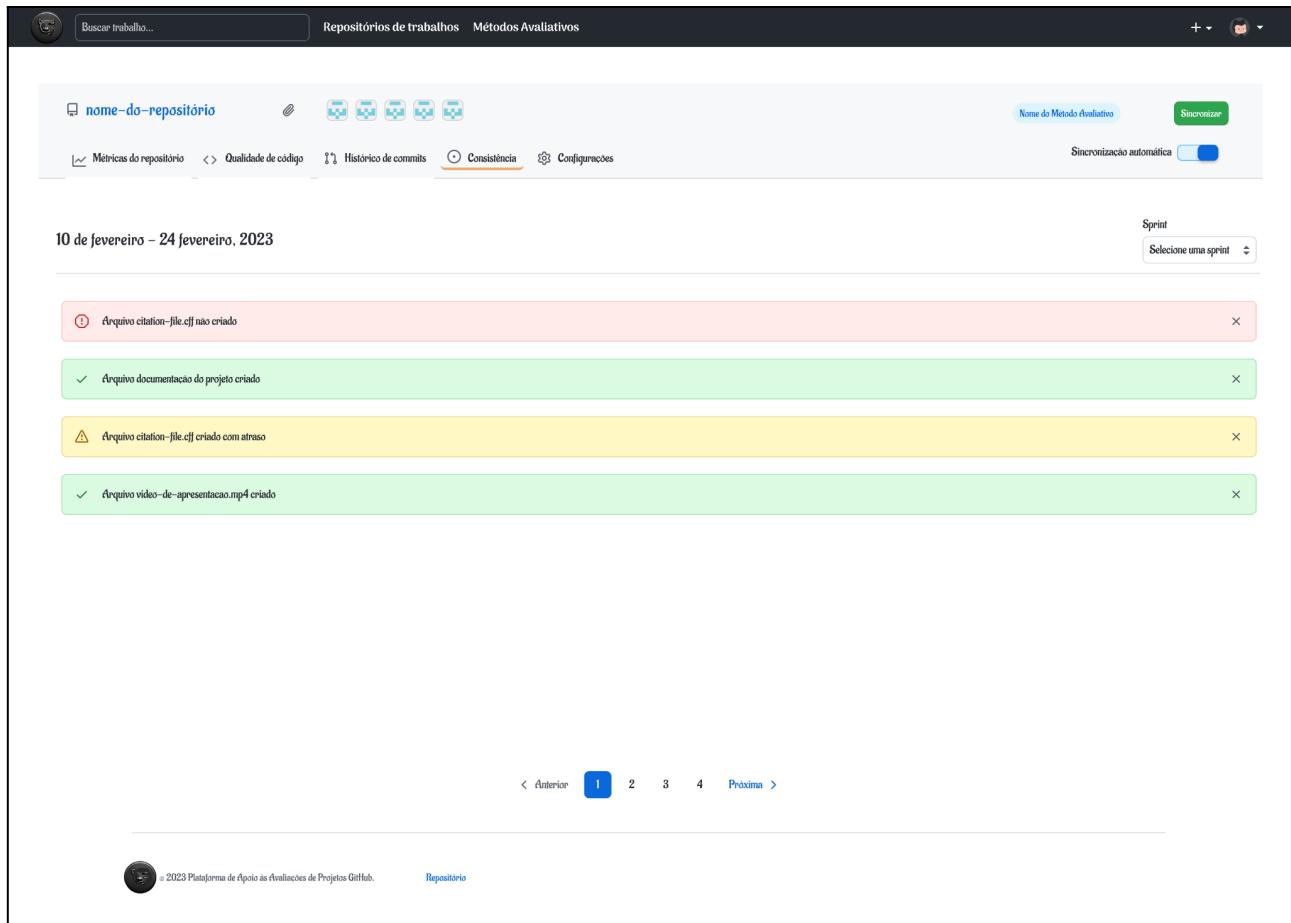


Figura 41. Tela da consistência de entrega de um trabalho

A Figura 42 apresenta a tela de visualização do trabalho, com a visualização de configurações do trabalho ativa. Nessa tela, é possível alterar a qual método avaliativo aquele trabalho pertence. Essa interface contempla o caso de uso UC4.

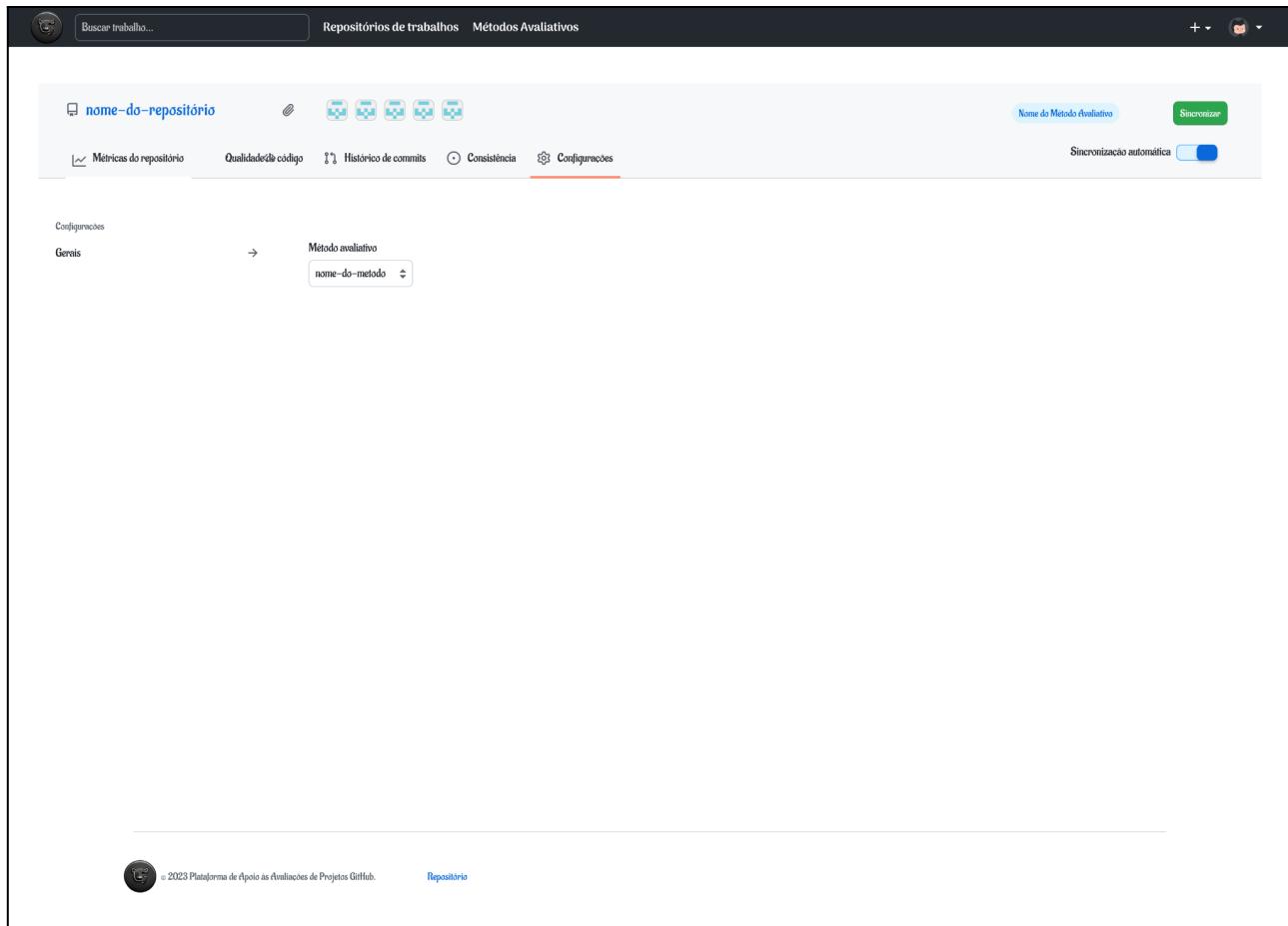


Figura 42. Tela de configurações de um trabalho

5. Glossário e Modelos de Dados

Esta seção tem o objetivo de apresentar e detalhar o glossário e modelo de dados do sistema, fornecendo uma compreensão mais abrangente sobre os termos técnicos utilizados e a estrutura de organização dos dados dentro da plataforma. Assim, a Tabela 28 especifica os atributos de entrada de dados no sistema, enquanto a Tabela 29 lista os atributos de saída de dados, permitindo uma compreensão mais clara das informações inseridas e geradas pela plataforma.

Autenticação na plataforma		
Atributo	Formato	Descrição
E-mail	Texto	E-mail do usuário do professor no GitHub.

Senha	Texto	Senha do usuário do professor no GitHub.
Gerenciar método avaliativo		
Descrição	Texto	Texto descritivo para caracterizar a disciplina à qual o método avaliativo se refere. Exemplo: “Trabalho Interdisciplinar 5: Aplicações Distribuídas”.
Semestre	Número	O semestre da oferta da disciplina do método avaliativo, sendo possível inserir 1 ou 2.
Ano	Número	O ano da oferta da disciplina do método avaliativo.
Desativado	Booleano	Indica se o método avaliativo está desativado ou não.
Gerenciar regra de consistência		
Descrição	Texto	Texto descritivo para caracterizar a qual é a regra de consistência. Exemplo: “Regra para arquivo de documentação do projeto”.
Diretório e nome do arquivo	Texto	Texto referente ao diretório e nome do arquivo da regra de consistência. Exemplo: “Documentacao/arquivo_de_arquitetura.md”.
Extensões de arquivo suportadas	Lista de texto	Lista de extensões de arquivos suportados pela regra de consistência. Exemplo: “md, pdf, docx”.
<i>Sprint</i> de entrega	<i>Sprint</i>	A <i>sprint</i> que esta regra de consistência deve ser entregue. Exemplo: “Sprint 1 - Trabalho Interdisciplinar 5: Aplicações Distribuídas 1/2023”.
<i>Issue</i> padronizada	<i>Issue</i> padronizada	A <i>issue</i> padronizada que será aberta no repositório dos trabalhos que não seguirem a

		regra de consistência delimitada para cada <i>sprint</i> . Exemplo: “Issue padronizada 1 - Faltando arquivo <i>Citation.cff</i> ”.
Gerenciar <i>sprint</i>		
Data do início	Data	Data do início da <i>sprint</i> . Exemplo: “01/08/2023”.
Data do fim	Data	Data do fim da <i>sprint</i> . Exemplo: “15/08/2023”.
Gerenciar <i>issue</i> padronizada		
Título da <i>issue</i> padronizada	Texto	Texto referente ao título da <i>issue</i> que será aberta nos repositórios de trabalho. Exemplo: “Faltando arquivo <i>Citation.cff</i> ”.
Descrição da <i>issue</i> padronizada	Texto	Texto referente à descrição da <i>issue</i> que será aberta nos repositórios de trabalho. Exemplo: “Preencha e envie o arquivo <i>Citation.cff</i> ”.

Tabela 28. Atributos de entrada de dados no sistema

Visualização de repositório de trabalho		
Atributo	Formato	Descrição
Nome do repositório	Texto	Texto referente ao nome do repositório de trabalho no GitHub.
<i>Link</i> do repositório	Texto	Texto do <i>link</i> do GitHub referente ao repositório do trabalho.
Método Avaliativo	Texto	Nome do método avaliativo que está avaliando o repositório.
<i>Branches</i>	Lista de texto	Nomes das <i>branches</i> que o repositório possui no GitHub.
<i>Commits</i>	Lista de texto	Nomes dos <i>commits</i> que o repositório possui que cada <i>branch</i> no GitHub.

Quantidade de linhas de código	Número	Número referente a quantidade de linhas que o repositório possui no GitHub.
Quantidade de arquivos	Número	Número referente a quantidade de arquivos que o repositório possui no GitHub.
Qualidade de código	Número	Número referente à qualidade do código calculado por meio do SonarQube para cada <i>commit</i> do repositório no GitHub.
Quantidade de <i>issues</i>	Número	Número de <i>issues</i> que o repositório possui no GitHub.
Quantidade de <i>pull requests</i>	Número	Número de <i>pull requests</i> que o repositório possui no GitHub.
Sincronização automática	Booleano	Indica se a sincronização diária de dados do repositório com o GitHub está ativada ou não.
Sincronizando	Booleano	Indica se o repositório está sincronizando os dados com o GitHub no momento ou não.
Alunos contribuidores	Número	Número de alunos que contribuíram com o repositório no GitHub.
Entrega de regra de consistência	Lista de entregas de regra de consistência	Refere-se a lista das entregas de regra de consistência entregues ou não por cada repositório do trabalho.
Visualização de método avaliativo		
Descrição	Texto	Texto descritivo para caracterizar a qual disciplina o método avaliativo refere-se. Exemplo: “Trabalho Interdisciplinar 5: Aplicações Distribuídas”.
Semestre	Número	O semestre da oferta da disciplina do método avaliativo, sendo possível inserir 1 ou 2.
Ano	Número	O ano da oferta da disciplina do

		método avaliativo.
Desativado	Booleano	Se o método avaliativo está desativado.
Repositórios	Lista de repositórios	Lista dos repositórios avaliados pelo método avaliativo.
Regras de consistência	Lista de regras de consistência	Lista das regras de consistência de um método avaliativo.
<i>Sprints</i>	Lista de <i>sprints</i>	Lista das <i>sprints</i> de um método avaliativo.
Issues Padronizadas	Lista de <i>issues</i> padronizadas	Lista das issues padronizadas de um método avaliativo.

Tabela 29. Atributos de saída de dados no sistema

A Figura 43 ilustra o Diagrama de Entidades e Relacionamentos (DER) do sistema, cujo objetivo é representar a camada de banco de dados da plataforma. Esse DER representa visualmente a estrutura do banco de dados do sistema e exibe um total de 18 tabelas, sendo 15 tabelas de entidades e 3 tabelas de relacionamento muitos-para-muitos. Esse diagrama foi modelado utilizando a ferramenta MySQL Workbench⁸. O símbolo de chave amarela define que aquela é a chave primária simples da tabela e a chave vermelha são chaves primárias compostas utilizadas em tabelas de associação muitos-para-muitos. Já o losango vermelho são as chaves estrangeiras, o losango azul preenchido referência atributos que não aceitam valores nulos e, por fim, o losango azul não preenchido são atributos que aceitam valores nulos. Ademais, esse modelo foi feito para possuir compatibilidade com o Sistema de Gerenciamento de Banco de Dados (SGBD) MySQL Server⁹ na versão 5.7.

⁸ <https://www.mysql.com/products/workbench/>

⁹ <https://dev.mysql.com/downloads/mysql/5.7.html>

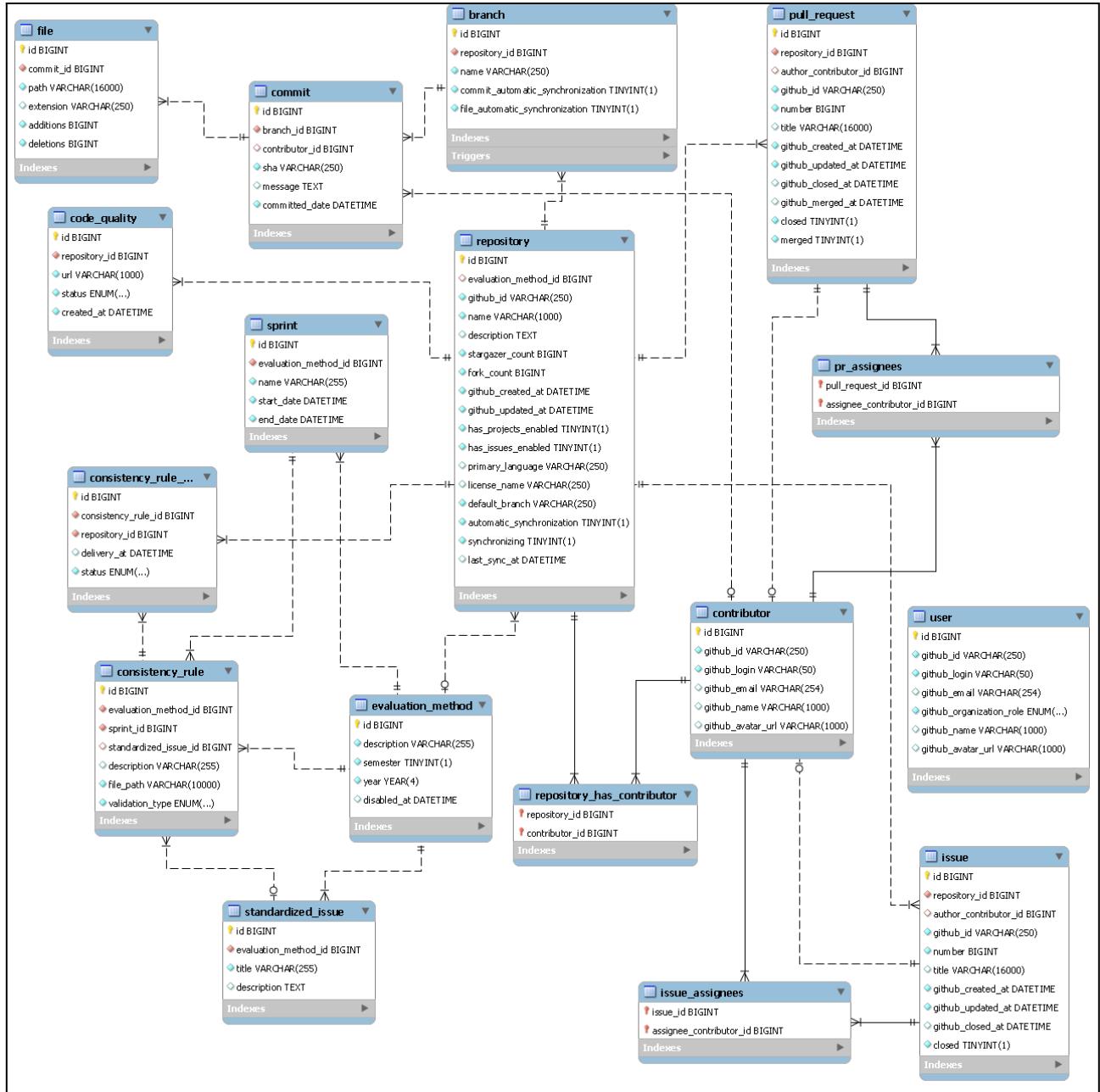


Figura 43. Diagrama de Entidade e Relacionamento

6. Casos de Teste

Esta seção apresenta casos de teste de aceitação e integração realizados conforme a especificação do sistema. Na Seção 6.1 são detalhados os casos de testes de aceitação, elaborados com base nas necessidades descritas no documento de visão do projeto, esses testes foram projetados para assegurar que as funcionalidades desenvolvidas atendam às necessidades dos usuários no sistema de forma satisfatória e confiável. Já na Seção 6.2 são descritos os casos de teste de integração

elaborados com o objetivo de analisar como o sistema se integra com os diversos componentes externos.

Esses testes foram planejados para garantir que todas as funcionalidades do sistema operem de maneira harmoniosa e confiável em conjunto com os componentes externos envolvidos. Cada caso de teste possui uma pré-condição, isso é, o estado no qual o sistema se encontra antes da execução do teste, além das ações que serão realizadas pelo usuário para interagir com o sistema durante a execução do teste. Por fim, é apresentado o resultado esperado, ou seja, o estado final que o sistema deverá apresentar após a execução do teste.

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 construídos com base nas necessidades (N) mapeadas no documento de visão. A seguir, são listadas essas necessidades, identificadas pela letra N e uma numeração única.

- N1. Autenticar usando GitHub;
- N2. Ver entregas de um trabalho;
- N3. Ver entregas de trabalho por *sprint*;
- N4. Ver contribuições de um integrante do trabalho;
- N5. Executar avaliação de qualidade de código do trabalho;
- N6. Ver métricas de contribuição do trabalho;
- N7. Abertura de *issues* padronizadas;
- N8. Detectar más práticas do Git nos trabalhos.

O funcionamento de cada caso de teste de aceitação é descrito a partir de tabelas. Essas tabelas são estruturadas com as seguintes colunas: identificador, necessidade, caso de teste, pré-condições necessárias, dados de entrada, ações e dado de resposta. O identificador é uma chave composta pela necessidade e o caso de teste que ele se refere. A necessidade é o nome da necessidade que aquele caso de teste valida. O caso de teste é um nome descritivo dado ao teste de aceitação para identificar o cenário de teste que está sendo executado. As pré-condições necessárias para o teste de aceitação são os requisitos que precisam ser atendidos antes da execução do teste. Os dados de entrada são as informações fornecidas ao sistema durante o teste. As ações são as etapas necessárias para realizar o teste de aceitação. O dado de resposta esperado ao final do teste de aceitação é a saída ou resposta que o sistema deve fornecer após a execução do teste.

A seguir são apresentadas os testes de aceitação relacionados à necessidade N1. A Tabela 30 apresenta o caso de teste de se autenticar corretamente na plataforma. Enquanto isso, a Tabela 31 apresenta o caso de teste de bloquear a autenticação para usuários que não possuem conta no GitHub. Por último, a Tabela 32 apresenta o caso de teste de bloquear a autenticação para usuários acesso à organização do GitHub.

Identificador	N1TI
---------------	------

Necessidade	Autenticar usando GitHub
Caso de teste	Autenticar na plataforma
Pré-condições	Possuir login no GitHub e acesso à organização.
Dados de entrada	E-mail e senha do usuário do GitHub (usuário com acesso à organização)
Ações	Clicar no botão do formulário de <i>login</i> do GitHub.
Dado de resposta	Usuário autenticado na plataforma.

Tabela 30. Teste de aceitação de autenticar na plataforma

Identificador	N1T2
Necessidade	Autenticar usando GitHub
Caso de teste	Não autenticar usuários inexistentes no GitHub
Pré-condições	Possuir login no GitHub sem acesso à organização.
Dados de entrada	E-mail e senha do usuário do GitHub (usuário inexistente)
Ações	Clicar no botão do formulário de <i>login</i> do GitHub.
Dado de resposta	Usuário é alertado de que não encontrou usuário, solicitando novamente a autenticação.

Tabela 31. Teste de aceitação de não autenticar usuários inexistentes

Identificador	N1T3
Necessidade	Autenticar usando GitHub
Caso de teste	Não autenticar usuários sem acesso à organização
Pré-condições	Nenhuma
Dados de entrada	E-mail e senha do usuário do GitHub (usuário sem acesso à organização)
Ações	Clicar no botão do formulário de <i>login</i> do GitHub.
Dado de resposta	Usuário é alertado de que não possui acesso à organização, solicitando novamente a autenticação.

Tabela 32. Teste de aceitação de não autenticar usuários sem acesso

A seguir são apresentadas os testes de aceitação relacionados à necessidade N2. A Tabela 33 apresenta o caso de teste de cadastrar uma nova regra de consistência em um método avaliativo, enquanto a Tabela 34 apresenta o caso de teste de exceção para a tentativa de criar uma regra de consistência sem preencher as informações necessárias. Enquanto isso, a Tabela 35 apresenta o caso de teste de vincular um repositório a um método avaliativo. Em seguida, Tabela 36 apresenta o caso de teste de ver quais entregas foram realizadas em determinado repositório. Por fim, a Tabela 37 apresenta o caso de teste de exceção para o caso do usuário tentar visualizar a consistência de um repositório que não possui método avaliativo vinculado a ele.

Identificador	N2T1
Necessidade	Ver entregas de um trabalho
Caso de teste	Cadastrar regra de consistência para ser entregue em um método avaliativo.
Pré-condições	<ul style="list-style-type: none"> • Usuário autenticado • Método avaliativo cadastrado.
Dados de entrada	Caminho do arquivo, <i>issue</i> personalizada e <i>sprint</i> de entrega.
Ações	<ul style="list-style-type: none"> • Entrar na página de um método avaliativo; • Clicar na seção de regras de consistência; • Clicar no botão de criar regra de consistência; • Preencher informações; • Enviar formulário.
Dado de resposta	Mensagem de sucesso, e nova regra de consistência aparece na lista.

Tabela 33. Teste de aceitação de cadastrar regra de consistência

Identificador	N2T2
Necessidade	Ver entregas de um trabalho
Caso de teste	Bloquear o cadastro de regra de consistência sem preenchimento de informações.
Pré-condições	<ul style="list-style-type: none"> • Usuário autenticado • Método avaliativo cadastrado.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> • Entrar na página de um método avaliativo; • Clicar na seção de regras de consistência; • Clicar no botão de criar regra de consistência;

	<ul style="list-style-type: none"> • Enviar formulário.
Dado de resposta	Mensagem de erro nos campos obrigatórios

Tabela 34. Teste de aceitação de bloquear o cadastro de regra de consistência sem preenchimento de informações

Identificador	N2T3
Necessidade	Ver entregas de um trabalho
Caso de teste	Vincular repositório a método avaliativo.
Pré-condições	<ul style="list-style-type: none"> • Usuário autenticado • Método avaliativo cadastrado. • Repositórios criados na organização do GitHub.
Dados de entrada	Seleção de repositório
Ações	<ul style="list-style-type: none"> • Entrar na página de um método avaliativo; • Clicar em adicionar repositório. • Selecionar repositório e enviar formulário.
Dado de resposta	Mensagem de sucesso, e o novo repositório aparece na lista.

Tabela 35. Teste de vincular repositório a método avaliativo

Identificador	N2T4
Necessidade	Ver entregas de um trabalho
Caso de teste	Ver entregas realizadas em um repositório
Pré-condições	<ul style="list-style-type: none"> • Usuário autenticado; • Repositório vinculado a método avaliativo; • Regras de consistência cadastradas para método avaliativo.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> • Entrar na página de um repositório; • Clicar na seção de consistência.
Dado de resposta	É apresentada lista com todas as regras de consistência para aquele repositório, cada um indicando se foi entregue e se houve atraso.

Tabela 36. Teste de aceitação de ver entregas de um repositório

Identificador	N2T5
Necessidade	Ver entregas de um trabalho
Caso de teste	Bloquear visualização de entregas realizadas em um repositório sem vínculo de método avaliativo
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositórios criados na organização do GitHub.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Clicar na seção de consistência.
Dado de resposta	É apresentada uma mensagem ao usuário informando que essa visualização só é possível para repositórios vinculados a algum método avaliativo.

Tabela 37. Teste de aceitação bloquear visualização de entregas em um repositório sem vínculo de método avaliativo

A seguir são apresentadas os testes de aceitação relacionados à necessidade N3. A Tabela 38 apresenta o caso de teste de cadastrar uma nova *sprint* em um método avaliativo, enquanto a Tabela 39 apresenta o caso de teste de exceção para a tentativa de criar uma *sprint* sem preencher as informações necessárias. Enquanto isso, a Tabela 40 apresenta o caso de teste de visualizar as *sprints* de um método avaliativo. Por último, a Tabela 41 apresenta o caso de teste de ver quais entregas foram realizadas em determinado repositório em uma determinada *sprint*.

Identificador	N3T1
Necessidade	Ver entregas de trabalho por <i>sprint</i>
Caso de teste	Cadastrar <i>sprint</i> em um método avaliativo
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Método avaliativo cadastrado;
Dados de entrada	Nome da <i>sprint</i> , data de início e data de término.
Ações	<ul style="list-style-type: none"> ● Entrar em página de um método avaliativo; ● Clicar em seção de <i>sprints</i>; ● Clicar no botão de criar <i>sprint</i>; ● Preencher informações; ● Enviar formulário.
Dado de resposta	Mensagem de sucesso, e a nova <i>sprint</i> aparece na lista.

Tabela 38. Teste de aceitação de cadastrar *sprint* de um método avaliativo

Identificador	N3T2
Necessidade	Ver entregas de trabalho por <i>sprint</i>
Caso de teste	Bloquear o cadastro de <i>sprint</i> sem preenchimento de informações.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Método avaliativo cadastrado;
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar em página de um método avaliativo; ● Clicar em seção de <i>sprints</i>; ● Clicar no botão de criar <i>sprint</i>; ● Enviar formulário.
Dado de resposta	Mensagem de erro nos campos obrigatórios

Tabela 39. Teste de aceitação de bloquear o cadastro de *sprint* sem preenchimento de informações

Identificador	N3T3
Necessidade	Ver entregas de trabalho por <i>sprint</i>
Caso de teste	Ver <i>sprints</i> de um método avaliativo
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Método avaliativo cadastrado;
Dados de entrada	Nome da <i>sprint</i> , data de início e data de término.
Ações	<ul style="list-style-type: none"> ● Entrar em página de um método avaliativo; ● Clicar em seção de <i>sprints</i>; ● Clicar no botão de criar <i>sprint</i>;
Dado de resposta	É apresentado a lista das <i>sprints</i> que foram cadastradas para aquele método avaliativo.

Tabela 40. Teste de aceitação de ver *sprints* de um método avaliativo

Identificador	N3T4
Necessidade	Ver entregas de trabalho por <i>sprint</i>

Caso de teste	Ver entregas de um trabalho filtradas por <i>sprint</i>
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório vinculado a método avaliativo; ● <i>Sprints</i> cadastradas para o método avaliativo; ● Regras de consistência cadastradas para método avaliativo.
Dados de entrada	<i>Sprint</i> para filtro.
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Clicar na seção de entregas. ● Escolher uma <i>sprint</i>.
Dado de resposta	É apresentada lista com todos as regras de consistência da <i>sprint</i> filtrada, para aquele repositório, cada um indicando se foi entregue e se houve atraso.

Tabela 41. Teste de aceitação de ver entregas filtradas por *sprint*

A seguir são apresentadas os testes de aceitação relacionados à necessidade N4. A Tabela 42 apresenta o caso de teste de visualizar os *commits* de um integrante de um trabalho. Enquanto isso, a Tabela 43 apresenta o caso de teste de visualizar os arquivos alterados por integrante de um trabalho. Por último, a Tabela 44 apresenta o caso de teste de ver as *issues* abertas e fechadas por integrante do trabalho.

Identificador	N4T1
Necessidade	Ver contribuições de um integrante do trabalho;
Caso de teste	Ver histórico de <i>commits</i> do integrante do trabalho.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório criado na organização do GitHub; ● Integrante do grupo realizou algum <i>commit</i> no repositório.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Clicar na seção de histórico de <i>commits</i>. ● Clicar em ícone de contribuidor para filtrar.
Dado de resposta	É apresentada a lista dos <i>commits</i> realizados por aquele contribuidor no repositório.

Tabela 42. Teste de aceitação de ver *commits* por contribuidor

Identificador	N4T2
Necessidade	Ver contribuições de um integrante do trabalho
Caso de teste	Ver arquivos alterados por integrante do trabalho.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório criado na organização do GitHub; ● Integrante do grupo realizou algum <i>commit</i> no repositório.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Na seção de métricas, clicar na sub-seção de contribuições de arquivos. ● Clicar em ícone de contribuidor para filtrar.
Dado de resposta	É apresentada a lista dos arquivos alterados, criados e deletados por aquele contribuidor no repositório.

Tabela 43. Teste de aceitação de ver arquivos manipulados por contribuidor

Identificador	N4T3
Necessidade	Ver contribuições de um integrante do trabalho
Caso de teste	Ver <i>issues</i> criadas por integrante do trabalho.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório criado na organização do GitHub.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Na seção de métricas, clicar na sub-seção de <i>issues</i>. ● Clicar em ícone de contribuidor para filtrar.
Dado de resposta	É apresentada um gráfico com o número de <i>issues</i> criadas por contribuidor selecionado

Tabela 44. Teste de aceitação de ver *issues* por contribuidor

A seguir são apresentadas os testes de aceitação relacionados à necessidade N5. A Tabela 45 apresenta o caso de teste de disparar uma avaliação de qualidade de código em um repositório. Enquanto isso, a Tabela 46 apresenta o caso de teste de visualizar as avaliações de qualidade de código executadas em um trabalho. Em seguida, a Tabela 47 apresenta o caso de teste de ver, em detalhes, uma avaliação de qualidade de código de repositório. Por fim, a Tabela 48 apresenta o caso de teste de exceção de exibir o erro quando a análise estática de código é mal-sucedida.

Identificador	N5T1
Necessidade	Executar avaliação de qualidade de código do trabalho
Caso de teste	Disparar uma avaliação de qualidade de código.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório criado na organização do GitHub.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Clicar na seção de qualidade de código; ● Clicar em executar uma nova análise.
Dado de resposta	Mensagem de sucesso, e a nova avaliação de qualidade de código aparece na lista.

Tabela 45. Teste de aceitação de disparar uma avaliação de qualidade de código

Identificador	N5T2
Necessidade	Executar avaliação de qualidade de código do trabalho
Caso de teste	Ver avaliações de qualidade de código executadas
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório criado na organização do GitHub.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Clicar na seção de qualidade de código.
Dado de resposta	Lista de avaliações de qualidade de código executadas no repositório, com o seu respectivo <i>status</i> .

Tabela 46. Teste de aceitação de ver avaliações de qualidade de um repositório

Identificador	N5T3
Necessidade	Executar avaliação de qualidade de código do trabalho
Caso de teste	Ver detalhes de avaliação de qualidade de código de repositório.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório criado na organização do GitHub;

	<ul style="list-style-type: none"> • Avaliação de qualidade de código previamente executada; • Avaliação de qualidade de código terminada.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> • Entrar na página de um repositório; • Clicar na seção de qualidade de código. • Selecionar uma avaliação de qualidade de código para exibir.
Dado de resposta	Métricas obtidas na análise de qualidade de código são exibidas na tela.

Tabela 47. Teste de aceitação de ver detalhes de avaliação de qualidade de um repositório

Identificador	N5T4
Necessidade	Executar avaliação de qualidade de código do trabalho
Caso de teste	Ver detalhes de avaliação mal sucedida de qualidade de código de repositório.
Pré-condições	<ul style="list-style-type: none"> • Usuário autenticado; • Repositório criado na organização do GitHub; • Avaliação de qualidade de código previamente executada; • Avaliação de qualidade de código terminada com falha.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> • Entrar na página de um repositório; • Clicar na seção de qualidade de código.
Dado de resposta	É apresentado o registro da avaliação de qualidade de código com erro.

Tabela 48. Teste de aceitação de ver detalhes de avaliação de qualidade mal sucedida

A seguir são apresentadas os testes de aceitação relacionados à necessidade N6. A Tabela 49 apresenta o caso de teste de visualizar as métricas de contribuição de *commits* do repositório. Enquanto isso, a Tabela 50 apresenta o caso de teste de ver métricas de contribuição de tipos de arquivos do repositório. Em seguida, a Tabela 51 apresenta o caso de teste de ver métricas de contribuição de linhas de código do repositório. Depois, a Tabela 52 apresenta o caso de teste de ver métricas de contribuição de arquivos do repositório. Por fim, a Tabela 53 apresenta o caso de teste de ver métricas de qualidade de descrição dos *commits*.

Identificador	N6T1
Necessidade	Ver métricas de contribuição do trabalho;

Caso de teste	Ver métricas de contribuição de <i>commits</i> do repositório.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Re却tório criado na organização do GitHub;
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um re却tório;
Dado de resposta	Métricas de contribuições de <i>commits</i> por contribuidor e geral são exibidas na tela.

Tabela 49. Teste de aceitação de ver contribuição de *commits* do trabalho

Identificador	N6T2
Necessidade	Ver métricas de contribuição do trabalho;
Caso de teste	Ver métricas de contribuição de tipos de arquivos do re却tório.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Re却atorio criado na organização do GitHub;
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um re却atorio; ● Na seção de métricas, clicar na sub-seção de tipos de arquivos contribuídos.
Dado de resposta	Métricas dos tipos de arquivos contribuídos mais contribuído são exibidas na tela.

Tabela 50. Teste de aceitação de ver contribuição de tipos de arquivos do trabalho

Identificador	N6T3
Necessidade	Ver métricas de contribuição do trabalho;
Caso de teste	Ver métricas de contribuição de linhas de código do re却atorio.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Re却atorio criado na organização do GitHub;
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um re却atorio; ● Na seção de métricas, clicar na sub-seção de contribuições de linhas de código.

Dado de resposta	Métricas das linhas de código por contribuidor e geral são exibidas na tela.
------------------	--

Tabela 51. Teste de aceitação de ver contribuição de linhas de código do trabalho

Identificador	N6T4
Necessidade	Ver métricas de contribuição do trabalho;
Caso de teste	Ver métricas de contribuição de arquivos do repositório.
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório criado na organização do GitHub;
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Na seção de métricas, clicar na sub-seção de contribuições de arquivo.
Dado de resposta	Métricas de contribuições de arquivos por contribuidor e geral são exibidas na tela.

Tabela 52. Teste de aceitação de ver contribuição de arquivos do trabalho

Identificador	N6T5
Necessidade	Ver métricas de contribuição do trabalho;
Caso de teste	Ver métricas de qualidade de descrição dos <i>commits</i>
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Repositório criado na organização do GitHub;
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um repositório; ● Na seção de métricas, clicar na sub-seção de contribuições de qualidade da descrição de <i>commits</i>.
Dado de resposta	Métricas de qualidade de descrição de <i>commits</i> por contribuidor e geral são exibidas na tela.

Tabela 53. Teste de aceitação de ver qualidade da descrição dos *commits*

A seguir são apresentadas os testes de aceitação relacionados à necessidade N7. A Tabela 54 apresenta o caso de teste de cadastrar uma nova *issue* padronizada em um método avaliativo,

enquanto a Tabela 55 apresenta o caso de teste de exceção para a tentativa de criar uma *issue* padronizada sem preencher as informações necessárias. Enquanto isso, a Tabela 56 apresenta o caso de teste de ver as *issues* padronizadas de um método avaliativo. Por último, a Tabela 57 apresenta o caso de teste de abrir uma ao detectar problema de consistência.

Identificador	N7T1
Necessidade	Abertura de <i>issues</i> padronizadas
Caso de teste	Cadastrar <i>issues</i> padronizadas em um método avaliativo.
Pré-condições	<ul style="list-style-type: none"> • Usuário autenticado; • Método avaliativo cadastrado;
Dados de entrada	Título e descrição da <i>issue</i> padronizada.
Ações	<ul style="list-style-type: none"> • Entrar na página de um método avaliativo; • Clicar na seção de <i>issues</i> padronizadas; • Clicar em criar <i>issues</i> padronizadas; • Preencher informações; • Enviar formulário.
Dado de resposta	Mensagem de sucesso, e a nova <i>issue</i> padronizada aparece na lista.

Tabela 54. Teste de aceitação de cadastrar uma *issue* padronizada

Identificador	N7T2
Necessidade	Abertura de <i>issues</i> padronizadas
Caso de teste	Bloquear o cadastro de <i>issues</i> padronizados sem preenchimento de informações.
Pré-condições	<ul style="list-style-type: none"> • Usuário autenticado; • Método avaliativo cadastrado;
Dados de entrada	-
Ações	<ul style="list-style-type: none"> • Entrar na página de um método avaliativo; • Clicar na seção de <i>issues</i> padronizadas; • Clicar em criar <i>issues</i> padronizadas; • Enviar formulário.
Dado de resposta	Mensagem de erro nos campos obrigatórios

Tabela 55. Teste de aceitação de bloquear o cadastro de *issues* padronizados sem preenchimento de informações.

Identificador	N7T3
Necessidade	Abertura de <i>issues</i> padronizadas
Caso de teste	Ver <i>issues</i> padronizadas de um método avaliativo
Pré-condições	<ul style="list-style-type: none"> ● Usuário autenticado; ● Método avaliativo cadastrado;
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Entrar na página de um método avaliativo; ● Clicar na seção de <i>issues</i> padronizadas.
Dado de resposta	Lista das <i>issues</i> padronizadas do método avaliativo.

Tabela 56. Teste de aceitação de ver *issues* padronizadas de um método avaliativo

Identificador	N7T4
Necessidade	Abertura de <i>issues</i> padronizadas
Caso de teste	<i>Issue</i> aberta ao detectar problema de consistência.
Pré-condições	<ul style="list-style-type: none"> ● <i>Issue</i> personalizada cadastrada. ● <i>Issue</i> personalizada vinculada a uma regra de consistência. ● Entrega da regra de consistência não foi feita no repositório
Dados de entrada	-
Ações	<ul style="list-style-type: none"> ● Sincronizar repositório
Dado de resposta	<i>Issue</i> será criada no repositório do GitHub

Tabela 57. Teste de aceitação de abertura de *issues* padronizadas

A seguir são apresentadas o teste de aceitação relacionado à necessidade N8. A Tabela 58 apresenta o caso de teste de detectar *force-push* de *commits*.

Identificador	N8T1
Necessidade	Detectar más práticas de Git e GitHub nos trabalhos
Caso de teste	Detectar <i>force-push</i> de <i>commits</i> .
Pré-condições	<ul style="list-style-type: none"> ● Repositório cadastrado na organização do GitHub.

	<ul style="list-style-type: none"> • Squashes com rebases utilizando de commits com force-push no repositório Git.
Dados de entrada	-
Ações	<ul style="list-style-type: none"> • Sincronizar repositório • Abrir aba de commits do repositório • Ativar filtro de commits afetados por force-push
Dado de resposta	São apresentados os commits possivelmente afetados por force-push

Tabela 58. Teste de aceitação de detectar *squashes* com *rebases* utilizando de *commits* com *force-push*

6.2 Testes de Integração

Esta seção apresenta os casos de teste de integração (TI) visam testar o funcionamento dos componentes internos do sistema durante a interação com o ambiente externo. O intuito é validar as funcionalidades do sistema que envolvem operações de entrada e saída de dados com sistemas externos. Para isso, as tabelas presentes nesta seção contam com colunas necessárias para detalhar o funcionamento de cada caso de teste de integração. As tabelas possuem as seguintes colunas: identificador, caso de teste, sistemas envolvidos no teste de integração, interface utilizada na comunicação, pré-condições necessárias, dados de entrada, ações para iniciar a integração e o dado de resposta esperado ao final do teste.

O identificador é um número sequencial utilizado para identificar cada caso de teste de integração de forma única. O caso de teste é um nome descritivo dado ao teste de integração para identificar o cenário de teste que está sendo executado. Os sistemas envolvidos no teste de integração são os sistemas externos que interagem durante o teste. A interface utilizada na comunicação é como os sistemas se comunicam durante o teste de integração. As pré-condições necessárias para o teste de integração são os requisitos que precisam ser atendidos antes da execução do teste. Os dados de entrada para efetuar a integração são as informações fornecidas ao sistema durante o teste. As ações para iniciar a integração são as etapas necessárias para iniciar o teste de integração. O dado de resposta esperado ao final do teste de integração é a saída ou resposta que o sistema deve fornecer após a execução do teste.

A Tabela 59 apresenta o caso de teste de integração com o objetivo de testar a funcionalidade de autenticação de usuários no sistema utilizando a API do GitHub para gerenciamento de dados de usuários. O identificador TI1, o caso de teste “Autenticar usuário com credenciais válidas” é o nome dado para identificar o cenário específico de teste, o sistema envolvido é GitHub, sendo o sistema externo utilizado para autenticação de usuários. A interface utilizada para a comunicação é uma requisição Protocolo de Transferência de Hipertexto (HTTP, do inglês *HyperText Transfer Protocol*), no formato Transferência de Estado Representacional (REST, do inglês *Representational State Transfer*) o qual é uma forma de comunicação com a API do GitHub. Não há nenhum requisito específico a ser atendido antes do teste ser executado, por ser o início da interação com a plataforma. Os dados de entrada necessários para a realização do teste são o e-mail e senha do usuário do GitHub, que serão utilizados para autenticação. As ações necessárias para a realização do teste consistem em clicar no botão do formulário de *login* do GitHub. O dado

de resposta esperado ao fim do teste é o *token* de acesso aos dados de usuário do GitHub, o qual é fornecido ao sistema interno após a autenticação do usuário no sistema externo. Este teste tem como finalidade validar a funcionalidade de autenticar o usuário no sistema externo do GitHub e verificar se a integração entre os sistemas ocorre de forma correta e eficiente.

Identificador	TI1
Caso de teste	Autenticar usuário com credenciais válidas
Sistemas envolvidos	GitHub
Interface	Requisição HTTP no formato REST
Pré-condições	Nenhuma
Dados de entrada	E-mail e senha do usuário do GitHub
Ações	Clicar no botão do formulário de <i>login</i> do GitHub
Dado de resposta	<i>Token</i> de acesso aos dados de usuário do GitHub

Tabela 59. Teste de integração de autenticar usuário com sucesso

A Tabela 60 apresenta o caso de teste de integração com o objetivo de testar a exceção de credenciais inválidas da funcionalidade de autenticação de usuários no sistema utilizando a API do GitHub. O identificador TI2, o caso de teste “Não autenticar usuário com credenciais inválidas” é o nome dado para identificar o cenário específico de teste, o sistema envolvido é GitHub, sendo o sistema externo utilizado para autenticação de usuários. A interface utilizada para a comunicação é uma requisição HTTP, no formato REST, o qual é uma forma de comunicação com a API do GitHub. Não há nenhum requisito específico a ser atendido antes do teste ser executado, por ser o início da interação com a plataforma. Os dados de entrada necessários para a realização do teste são o e-mail ou senha inválidos do usuário do GitHub, que serão utilizados para autenticação. As ações necessárias para a realização do teste consistem em clicar no botão do formulário de *login* do GitHub. O dado de resposta esperado ao fim do teste é autenticação falhou devido a credenciais inválidas. Este teste tem como finalidade validar a exceção funcionalidade de autenticar o usuário no sistema externo do GitHub quando um e-mail ou senha incorretos são inseridos.

Identificador	TI2
Caso de teste	Não autenticar usuário com credenciais inválidas
Sistemas envolvidos	GitHub
Interface	Requisição HTTP no formato REST
Pré-condições	Nenhuma

Dados de entrada	E-mail ou senha incorretos do usuário do GitHub
Ações	Clicar no botão do formulário de <i>login</i> do GitHub
Dado de resposta	Apresentar que a autenticação falhou por credenciais inválidas

Tabela 60. Teste de integração de autenticar usuário com credenciais inválidas

A Tabela 61 apresenta o caso de teste de integração com o objetivo de testar a exceção de usuário não administrativo da funcionalidade de autenticação de usuários no sistema utilizando a API do GitHub. O identificador TI3, o caso de teste “Não autenticar usuário com credenciais não administrativas da organização do GitHub” é o nome dado para identificar o cenário específico de teste, o sistema envolvido é GitHub, sendo o sistema externo utilizado para autenticação de usuários. A interface utilizada para a comunicação é uma requisição HTTP, no formato REST, o qual é uma forma de comunicação com a API do GitHub. Não há nenhum requisito específico a ser atendido antes do teste ser executado, por ser o início da interação com a plataforma. Os dados de entrada necessários para a realização do teste são o e-mail ou senha do usuário não administrativo da organização do GitHub, que serão utilizados para autenticação. As ações necessárias para a realização do teste consistem em clicar no botão do formulário de *login* do GitHub. O dado de resposta esperado ao fim do teste é autenticação falhou devido de usuário não administrativo. Este teste tem como finalidade validar a exceção funcionalidade de autenticar o usuário no sistema externo do GitHub quando o e-mail e senha é de um usuário não administrativo da organização.

Identificador	TI3
Caso de teste	Não autenticar usuário com credenciais não administrativas da organização do GitHub
Sistemas envolvidos	GitHub
Interface	Requisição HTTP no formato REST
Pré-condições	Nenhuma
Dados de entrada	E-mail e senha de usuário não administrativo
Ações	Clicar no botão do formulário de <i>login</i> do GitHub
Dado de resposta	Apresentar que a autenticação falhou por credenciais não administrativas

Tabela 61. Teste de integração de autenticar usuário com credenciais não administrativas da organização

A Tabela 62 apresenta o caso de teste de integração com o objetivo de testar a funcionalidade de buscar dados de usuário logado no sistema utilizando a API do GitHub para gerenciamento de dados de usuários. O identificador é TI4 e o caso de teste é “Buscar dado de usuário logado”. O sistema envolvido é o GitHub, sendo o sistema externo utilizado para busca dos

dados de usuários. A interface utilizada para a comunicação é uma requisição HTTP no formato REST que é uma forma de comunicação com a API do GitHub. A pré-condição necessária para a realização deste teste é que o usuário esteja autenticado, uma vez que o teste se trata de buscar dados do usuário logado. O dado de entrada necessário para a realização do teste é o *token* do usuário autenticado, utilizado para realizar a busca dos dados no GitHub. As ações necessárias para a realização do teste consistem no usuário acessar a página inicial da plataforma. O dado de resposta esperado ao fim do teste é que os dados da conta do GitHub do usuário autenticado sejam retornados corretamente para o sistema interno do usuário. Este teste tem como finalidade validar a funcionalidade de buscar dados do usuário autenticado no sistema externo do GitHub e verificar se a integração entre os sistemas ocorre de forma correta e eficiente.

Identificador	TI4
Caso de teste	Buscar dado de usuário logado
Sistemas envolvidos	GitHub
Interface	Requisição HTTP no formato REST
Pré-condições	O usuário estar autenticado.
Dados de entrada	O <i>token</i> do usuário autenticado
Ações	O usuário acessar a página inicial da plataforma
Dado de resposta	Dados da conta do GitHub do usuário autenticado

Tabela 62. Teste de integração de buscar dado de usuário logado

A Tabela 63 apresenta o caso de teste de integração com o objetivo de testar a exceção de buscar dados de usuário logado no sistema utilizando a API do GitHub para gerenciamento de dados de usuários, mas com o *token* de autenticação expirado. O identificador é TI5 e o caso de teste é “Buscar dado de usuário logado com token expirado”. O sistema envolvido é o GitHub, sendo o sistema externo utilizado para busca dos dados de usuários. A interface utilizada para a comunicação é uma requisição HTTP no formato REST que é uma forma de comunicação com a API do GitHub. A pré-condição necessária para a realização deste teste é que o usuário esteja autenticado, mas o seu *token* de autenticação tenha expirado. Os dados de entrada necessários para a realização do teste são o token do usuário autenticado, utilizado para realizar a busca dos dados no GitHub. As ações necessárias para a realização do teste consistem no usuário acessar a página inicial da plataforma. O dado de resposta esperado ao fim do teste é que seja alertado que a autenticação do usuário está expirada, além de fazer o redirecionamento para uma nova autenticação.

Identificador	TI5
Caso de teste	Buscar dado de usuário logado com <i>token</i> expirado

Sistemas envolvidos	GitHub
Interface	Requisição HTTP no formato REST
Pré-condições	O usuário ter autenticado e expirado a autenticação
Dados de entrada	O <i>token</i> do usuário autenticado
Ações	O usuário acessar alguma página da plataforma
Dado de resposta	Usuário expirado e redirecionado para tela de autenticação

Tabela 63. Teste de integração de buscar dado de usuário logado com *token* expirado

A Tabela 64 apresenta o caso de teste de integração com o objetivo de testar a funcionalidade de sincronização de dados de repositórios da organização no sistema utilizando a API do GitHub para gerenciamento de dados de repositórios de trabalhos. O identificador TI6, o caso de teste “Sincronizar dados de repositórios da organização com sucesso” é o nome dado para identificar o cenário específico de teste, o sistema envolvido é GitHub, sendo a interface de comunicação uma requisição HTTP, no formato REST por meio da biblioteca Octokit¹⁰. A pré-condição para este caso de teste é que o usuário já esteja autenticado no sistema. Não há dados de entrada necessários para a realização deste teste. As ações necessárias para a realização do teste consistem em selecionar um repositório de trabalho e clicar no botão de “Sincronizar”. O dado de resposta esperado ao fim do teste são as informações do repositório do GitHub sincronizadas com a plataforma. Este caso de teste é importante para garantir que as informações de repositórios da organização estejam corretamente integradas e sincronizadas entre o GitHub e a plataforma.

Identificador	TI6
Caso de teste	Sincronizar dados de repositórios da organização com sucesso
Sistemas envolvidos	GitHub
Interface	Requisição HTTP no formato REST por meio da biblioteca Octokit
Pré-condições	O usuário estar autenticado
Dados de entrada	Nenhum
Ações	Selecionar um repositório de trabalho e clicar no botão de “Sincronizar”
Dado de resposta	As informações do repositório do GitHub sincronizadas com a plataforma

Tabela 64. Teste de integração de sincronizar dados de repositórios da organização com sucesso

¹⁰ <https://octokit.github.io/rest.js/v20>

A Tabela 65 apresenta o caso de teste de integração com o objetivo de testar exceção da funcionalidade de sincronização de dados de repositórios da organização no sistema utilizando a API do GitHub para gerenciamento de dados de repositórios de trabalhos. O identificador TI7, o caso de teste “Sincronizar dados de repositórios da organização com falha” é o nome dado para identificar o cenário específico de teste, o sistema envolvido é GitHub, sendo a interface de comunicação uma requisição HTTP, no formato REST por meio da biblioteca Octokit. A pré-condição para este caso de teste é que o usuário já esteja autenticado no sistema. Não há dados de entrada necessários para a realização deste teste. As ações necessárias para a realização do teste consistem em selecionar um repositório de trabalho e clicar no botão “Sincronizar”. O dado de resposta esperado ao fim do teste é um aviso de falha ao sincronizar as informações do repositório do GitHub com a plataforma e uma nova tentativa será automaticamente efetuada em breve. Este caso de teste é importante para testar a probabilidade de falha na sincronização das informações de repositórios da organização entre o GitHub e a plataforma.

Identificador	TI7
Caso de teste	Sincronizar dados de repositórios da organização com falha
Sistemas envolvidos	GitHub
Interface	Requisição HTTP no formato REST por meio da biblioteca Octokit
Pré-condições	O usuário estar autenticado
Dados de entrada	Nenhum
Ações	Selecionar um repositório de trabalho e clicar no botão “Sincronizar”
Dado de resposta	Falha ao sincronizar e avisar que uma nova tentativa será executada automaticamente

Tabela 65. Teste de integração de sincronizar dados de repositórios da organização com falha

A Tabela 66 apresenta o caso de teste de integração com o objetivo de realizar uma análise estática de código de um repositório utilizando o SonarQube. O identificador TI8, o caso de teste “Realizar análise estática de código com sucesso utilizando SonarQube” é o nome dado para identificar este cenário específico de teste. Os sistemas envolvidos são o SonarQube e a API da plataforma. A interface de comunicação é uma requisição HTTP no formato REST, utilizando a API para executar a análise do SonarQube. As pré-condições para este caso de teste incluem o usuário estar autenticado na plataforma e o repositório de código estar sincronizado na plataforma. Não há dados de entrada necessários para a realização deste teste. As ações necessárias para a realização do teste incluem selecionar o repositório na plataforma, escolher a opção do menu “Qualidade de código” e clicar no botão “Executar análise de código”. O dado de resposta esperado ao fim do teste é o *status* “Análise concluída”, um relatório de análise estática gerado pelo SonarQube, detalhando métricas de qualidade, potenciais vulnerabilidades, bugs e débito técnico, o qual será acessível pelo *link* apresentado na interface da plataforma.

Identificador	TI8
Caso de teste	Realizar análise estática de código com sucesso utilizando SonarQube
Sistemas envolvidos	SonarQube e a API da plataforma
Interface	Requisição HTTP no formato REST, utilizando a API para executar a análise do SonarQube
Pré-condições	O usuário estar autenticado e o repositório de código estar sincronizado na plataforma
Dados de entrada	Nenhum
Ações	Selecionar o repositório na plataforma, escolher a opção do menu “Qualidade de código” e clicar no botão “Executar análise de código”
Dado de resposta	O <i>status</i> “Análise concluída”, um relatório de análise estática gerado pelo SonarQube, detalhando métricas de qualidade, potenciais vulnerabilidades, bugs e débito técnico, o qual será acessível pelo link apresentado na interface da plataforma

Tabela 66. Teste de integração de realizar análise estática de código com sucesso utilizando SonarQube

A Tabela 67 apresenta o caso de teste de integração com o objetivo de testar exceção da funcionalidade de realizar uma análise estática de código de um repositório utilizando o SonarQube. O identificador TI9, o caso de teste “Realizar análise estática de código com falha utilizando SonarQube” é o nome dado para identificar este cenário específico de teste. Os sistemas envolvidos são o SonarQube e a API da plataforma. A interface de comunicação é uma requisição HTTP no formato REST, utilizando a API para executar a análise do SonarQube. As pré-condições para este caso de teste incluem o usuário estar autenticado na plataforma e o repositório de código estar sincronizado na plataforma. Não há dados de entrada necessários para a realização deste teste. As ações necessárias para a realização do teste incluem selecionar o repositório na plataforma, escolher a opção do menu “Qualidade de código” e clicar no botão “Executar análise de código”. O dado de resposta esperado ao fim do teste é o *status* “Erro na análise” e um relatório de análise estática gerado pelo SonarQube com a falha, o qual será acessível pelo *link* apresentado na interface da plataforma.

Identificador	TI9
Caso de teste	Realizar análise estática de código com falha utilizando SonarQube
Sistemas envolvidos	SonarQube e a API da plataforma
Interface	Requisição HTTP no formato REST, utilizando a API para executar a

	análise do SonarQube
Pré-condições	O usuário estar autenticado e o repositório de código estar sincronizado na plataforma
Dados de entrada	Nenhum
Ações	Selecionar o repositório na plataforma, escolher a opção do menu “Qualidade de código” e clicar no botão “Executar análise de código”
Dado de resposta	O <i>status</i> “Erro na análise” e um relatório de análise estática gerado pelo SonarQube com a falha, o qual será acessível pelo <i>link</i> apresentado na interface da plataforma

Tabela 67. Teste de integração de realizar análise estática de código com falha utilizando SonarQube

7. Riscos

Nesta seção são apresentados os riscos de desenvolvimento deste projeto, visando destacar estratégias de mitigação para os possíveis problemas que possam surgir. Esses riscos são principalmente resultados da escala do sistema projetado e da alta dependência de sistemas.

O projeto envolve a implementação de 42 casos de uso e 18 tabelas de dados, tornando a codificação extensa. Um caso de uso em particular, o UC31, apresenta uma alta complexidade devido à sincronização de toda a base de dados da plataforma com o sistema do GitHub. A implementação dessa sincronização realizada com cuidado, uma vez que envolve a persistência em 10 tabelas de dados e as principais funcionalidades da plataforma, como métodos avaliativos e métricas dos trabalhos, dependem diretamente da sincronização dos repositórios. Além disso, a sincronização de bases de dados diferentes requer o tratamento temporal de informações, considerando os fusos horários, já que o GitHub é um sistema internacional. Portanto, é evidente a alta sensibilidade desse caso de uso, tornando necessário priorizar seu desenvolvimento na primeira *sprint* e implementar testes unitários e de integração para garantir a consistência.

Outros casos de uso que demandam um esforço significativo de desenvolvimento estão relacionados às operações de criação, leitura, atualização e deleção (CRUD, do acrônimo em inglês para *Create, Read, Update and Delete*) de várias entidades. Embora essas operações não sejam complexas em si, sua quantidade e o volume de dados envolvidos exigem atenção. Entre as operações de CRUD mais relevantes para o funcionamento da plataforma estão aquelas relacionadas a métodos avaliativos, regras de consistência, *sprints* e *issues* padronizadas. É importante ressaltar que a funcionalidade de *issue* padronizada não faz parte do mínimo necessário a ser desenvolvido, pois a plataforma pode operar tanto com quanto sem ela.

Considerando os riscos relacionados à integração com sistemas externos, o GitHub e o SonarQube estão envolvidos. O UC1 refere-se à autenticação dos usuários na plataforma por meio do GitHub, sendo necessário validar se o usuário é um administrador da organização ICEI-PUC-Minas-PPLES-TI. Nesse sentido, a API do GitHub deve fornecer as informações necessárias para realizar essa validação. Caso essas informações não sejam fornecidas, é necessário mitigar esse problema implementando um sistema de autenticação alternativo. Como essas

informações estão presentes na API do GitHub, foi possível implementar a autenticação com OAUTH diretamente do GitHub. Outro risco a ser mitigado em relação ao GitHub é o consumo massivo de dados durante a sincronização dos repositórios, uma vez que há um limite de requisições à API REST por hora. Portanto, é necessário definir um máximo de sincronização automática para cada repositório por dia.

No que diz respeito à integração com a ferramenta SonarQube, visa-se utilizar métricas básicas disponíveis na interface pública do SonarQube, a fim de evitar aumentar a complexidade dessa funcionalidade. No caso de haver restrições significativas na visualização de dados por meio das interfaces públicas de codificação, é possível considerar a disponibilização de um *link* direto para a página de análise gerada pelo SonarQube para o repositório de trabalho, como uma medida de mitigação. É importante ressaltar que, se essa mitigação não for possível, essa integração, embora seja útil, não afetará o funcionamento das demais funcionalidades, caso não possa ser implementada. Durante a implementação, a abordagem mais apropriada foi a utilização do *link* direto, dessa forma, pôde-se aproveitar de todas as métricas providas pelo SonarQube, sem nenhuma restrição ou necessidade de re-implementação dos indicadores.

8. Cronograma e Processo de Implementação

Nesta seção é apresentado o cronograma de desenvolvimento do trabalho e o processo de implementação utilizado. A Seção 8.1 detalha o cronograma geral do trabalho, com a divisão das entregas por meio dos *milestones* do GitHub que possuem os prazos de entrega para as *issues* associadas. Além disso, também na Seção 8.1 é apresentado um cronograma de entregas para às 8 *sprints* quinzenais de desenvolvimento da plataforma. Na Seção 8.2, são detalhadas as práticas, metodologias, estratégias e artefatos produzidos durante o processo de implementação.

8.1 Cronograma

O cronograma de desenvolvimento deste trabalho está dividido em duas fases: documentação do projeto e desenvolvimento de código da plataforma. A fase de documentação do projeto compreende as entregas do projeto no período de 12 de fevereiro de 2023 até 16 de junho de 2023, enquanto a fase de desenvolvimento de código ocorre no período de 15 de agosto de 2023 até 30 de novembro de 2023.

A Figura 44 apresenta 6 *milestones* do GitHub relacionadas às entregas de documentação do projeto. A primeira *milestone*, intitulada “TCC 1 - Entrega A1”, contempla a entrega do artefato Memorial Descritivo da Graduação. Na segunda *milestone*, “TCC 1 - Entrega A2”, foram entregues os artefatos Documento de Visão e Estudo de Viabilidade. Já na terceira, quarta e quinta *milestones*, “TCC 1 - Entrega A3”, “TCC 1 - Entrega A4” e “TCC 1 - Entrega A5”, respectivamente, foram entregues seções do artefato Documento do Projeto. As *issues* foram associadas as *milestones* no GitHub para divisão das tarefas ou problemas específicos que precisam ser resolvidos para entrega estabelecida pela *milestone*. As *issues* foram atribuídas aos alunos Guilherme Gabriel Silva Pereira e Lucas Ângelo Oliveira Martins Rocha e marcadas com *labels* que ajudam a classificar e organizar as *issues*, como, por exemplo, tarefas de documentação, *design*, codificação entre outras. As

milestones e *issues* são uma forma eficaz de acompanhar o progresso do projeto e garantir que todos os itens necessários para a entrega sejam concluídos dentro do prazo estabelecido.

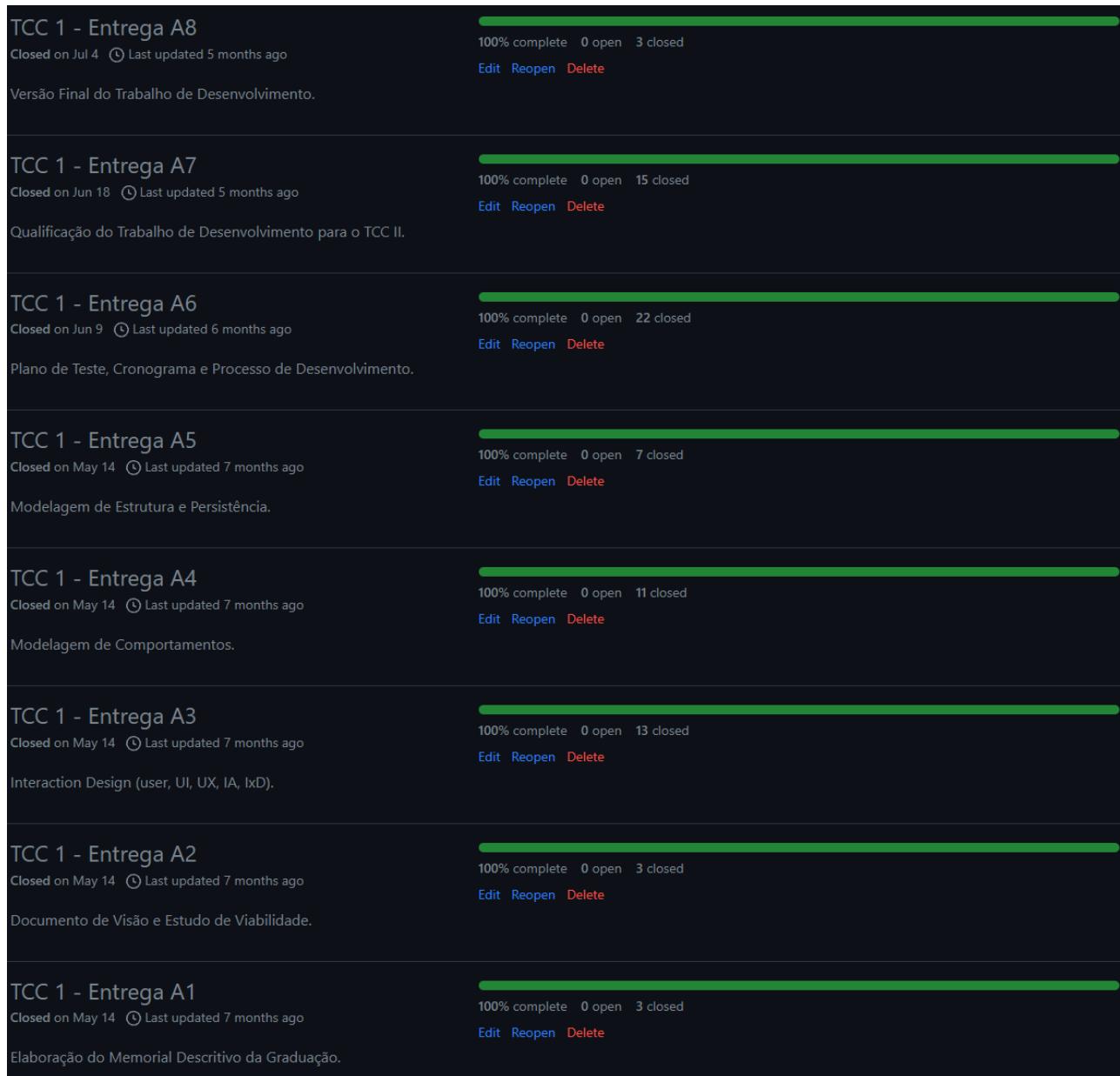


Figura 44. Milestones das entregas de documentação do projeto

Na Figura 45, apresenta-se o conjunto de 8 *milestones*, correspondente a cada *sprint* na fase de desenvolvimento da plataforma. Durante o processo de desenvolvimento da plataforma, foram abertas *issues* correspondentes a cada entrega de código necessária para a *sprint* correspondente. Essas *issues* foram criadas para cada *milestone* do projeto, garantindo que cada etapa esteja devidamente documentada e rastreada. Cada *issue* foi atribuída a um ou mais participantes responsáveis pela sua implementação, que trabalharam em conjunto para entregá-la dentro do prazo estabelecido, pelo fato de algumas entregas de código serem conjuntas.

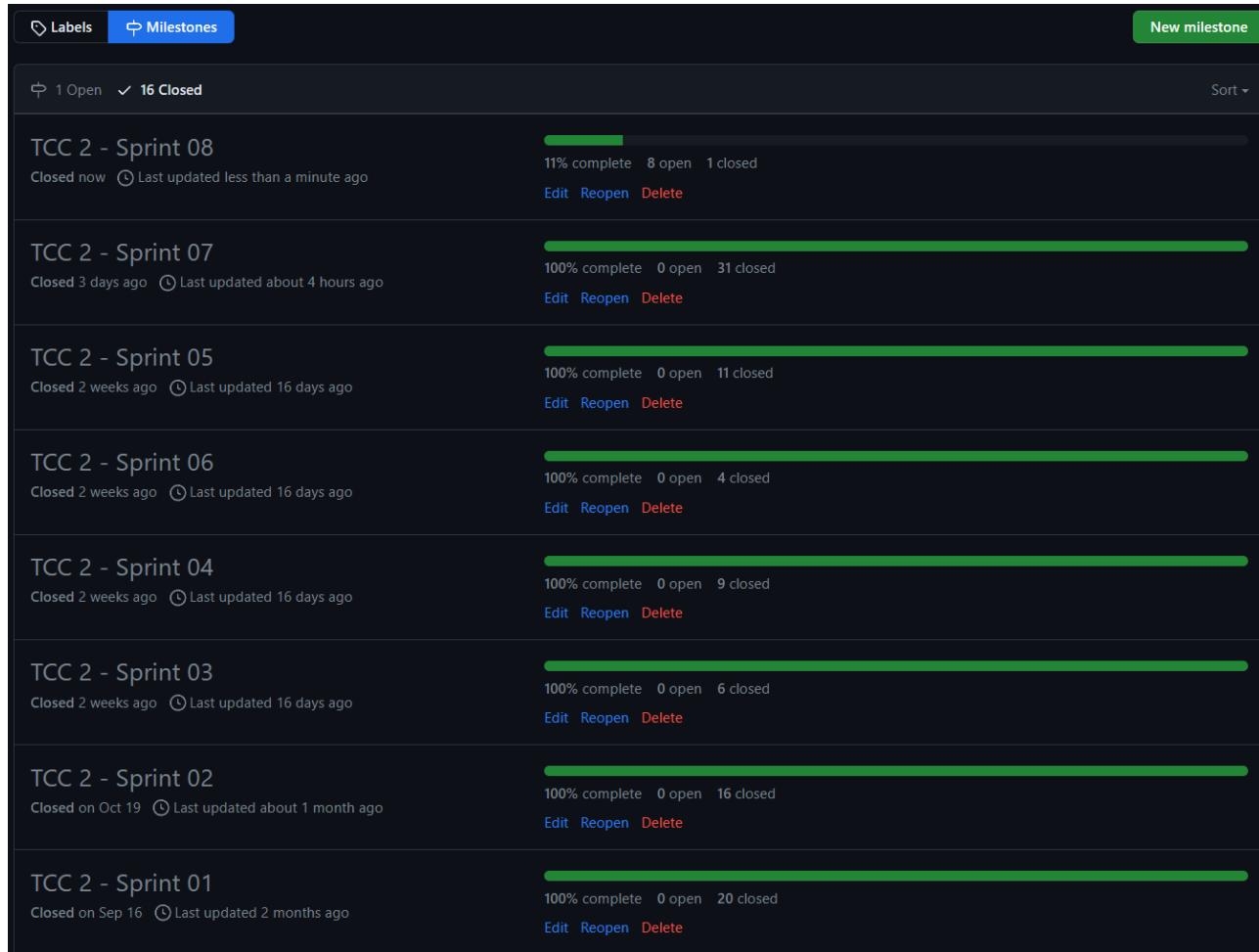


Figura 45. Milestones das sprints de desenvolvimento de código da plataforma

Na Tabela 68, apresenta-se o cronograma de desenvolvimento da plataforma, dividido em 8 sprints com duração de quinze ou dezesseis dias, ocorrendo entre os meses de agosto e novembro de 2023. A coluna “Atividades” foi dividida entre os alunos participantes do projeto, a fim de esclarecer quais são as atividades executadas por cada um em cada período. As atividades foram levantadas com base nas necessidades da aplicação e na documentação do sistema previamente elaborada. Além disso, para cada *sprint*, foram abertas *issues* correspondentes às entregas de código necessárias para a realização das tarefas estipuladas. Dessa forma, cada participante pode visualizar e executar as atividades atribuídas a ele, conforme as *issues* associadas a cada *milestone*.

Nome	Período	Atividades	
		Guilherme Gabriel	Lucas Ângelo
Sprint 1	01/08/2023 –	<ul style="list-style-type: none"> • Criar estrutura de <i>backend</i> do SupportPlatform-Service: 	<ul style="list-style-type: none"> • Criar infraestrutura de <i>devops</i> do SupportPlatform-Service:

	15/08/2023 (15 dias)	<ul style="list-style-type: none"> ○ Configurar diretórios, bibliotecas e variáveis de ambiente; ○ Configurar documentação das rotas da API; ○ Adicionar configuração de testes de integração da API. ● Criar estrutura do <i>frontend</i> do SupportPlatform-Web: <ul style="list-style-type: none"> ○ Configurar diretórios, bibliotecas e variáveis de ambiente; ○ Configurar sistema de rotas da aplicação; ○ Configurar sistema de comunicação com a API e query dos dados. ● Implementar interface de lista de repositórios; ● Configurar estrutura de controle de qualidade de <i>commits</i> usado <i>hooks do Git</i>: <ul style="list-style-type: none"> ○ Análise estática de código antes de permitir <i>commit</i>; ○ Análise da mensagem de <i>commit</i> para garantir que este segue o padrão. ● Implementar <i>backend</i> da listagem de repositórios da SupportPlatform-Service; ● Casos de uso relacionado: UC2. 	<ul style="list-style-type: none"> ○ Configurar o Docker para implantação; ○ Configurar GitHub Action para integração Contínua. ● Criar infraestrutura de <i>devops</i> do SupportPlatform-Web: <ul style="list-style-type: none"> ○ Configurar Docker para implantação; ○ Configurar GitHub Action para integração contínua. ● Criar infraestrutura de <i>devops</i> do JobScheduler-Service: <ul style="list-style-type: none"> ○ Configurar o Docker para implantação; ○ Configurar GitHub Action para integração contínua. ● Criar estrutura de <i>backend</i> do JobScheduler-Service: <ul style="list-style-type: none"> ○ Configurar diretórios, bibliotecas e variáveis de ambiente. ● Implementar sincronização de repositórios no JobScheduler-Service: <ul style="list-style-type: none"> ○ Configurar conexão com banco de dados; ○ Codificar modelos, interfaces, serviços, utilitários, registros (Do inglês, <i>logs</i>), variáveis de ambiente e buscadores de dados; ○ Definir integração com GitHub por meio de autenticação e funções do Octokit; ○ Mapear entidades <i>Repository</i>, <i>Contributor</i>, <i>PullRequest</i>, <i>Issue</i>,
--	-------------------------	---	--

			<p><i>Branch, Commit e File</i> do GitHub para o banco de dados;</p> <ul style="list-style-type: none"> ○ Implementar sincronização em massa de dados do GitHub com o banco de dados; ○ Codificar <i>cronjob</i> para sincronização automática. <ul style="list-style-type: none"> ● Casos de uso relacionados: UC31 e UC32; ● Testes de Integração relacionados: TI6 e TI7.
Sprint 2	16/08/2023 – 31/08/2023 (16 dias)	<ul style="list-style-type: none"> ● Implementar <i>backend</i> de métricas de repositório da SupportPlatform-Service: <ul style="list-style-type: none"> ○ Quantidade e porcentagem de <i>commits</i>; ○ Quantidade de linhas alteradas. ● Implementar <i>backend</i> de ver repositório da SupportPlatform-Service. ● Implementar interface de repositório; ● Casos de uso relacionados: UC15, UC16, e UC32; ● Testes de Aceitação relacionados: N6T1 e N6T3. 	<ul style="list-style-type: none"> ● Desenvolver <i>backend</i> da autenticação integrada com o GitHub no SupportPlatform-Service; ● Implementar <i>backend</i> de métodos avaliativos no SupportPlatform-Service: <ul style="list-style-type: none"> ○ Criação, atualização, leitura e deleção; ○ Testes unitários e de integração. ● Implementar <i>backend</i> de <i>sprints</i> no SupportPlatform-Service: <ul style="list-style-type: none"> ○ Criação, atualização, leitura e deleção; ○ Testes unitários e de integração. ● Casos de uso relacionados: UC1, UC3, UC4, UC5, UC21, UC27, UC33, UC34, UC35, UC37 e UC38.
Sprint 3	01/09/2023 – 15/09/2023 (15 dias)	<ul style="list-style-type: none"> ● Implementar <i>backend</i> de métricas de repositório da SupportPlatform-Service: <ul style="list-style-type: none"> ○ Tipos de arquivos contribuídos; ○ Contribuições em 	<ul style="list-style-type: none"> ● Implementar <i>backend</i> de regras de consistência no SupportPlatform-Service: <ul style="list-style-type: none"> ○ Criação, atualização, leitura e deleção; ○ Testes unitários e de

		<ul style="list-style-type: none"> ○ <i>issues</i>; ○ Qualidade da descrição dos <i>commits</i>. ● Implementar interface de métricas de repositório <ul style="list-style-type: none"> ○ Quantidade e porcentagem de <i>commits</i>; ○ Linhas alteradas; ○ Tipos de arquivos contribuídos; ○ Contribuições em <i>issues</i>; ○ Qualidade da descrição dos <i>commits</i>. ● Casos de uso relacionados: UC15, UC16, UC17, UC19 e UC20; ● Testes de Aceitação relacionados: N4T3, N6T1, N6T2, N6T3, N6T5. 	<p>integração.</p> <ul style="list-style-type: none"> ● Implementar <i>backend</i> de entrega de regra de consistência no SupportPlatform-Service: <ul style="list-style-type: none"> ○ Criação, atualização, leitura e deleção; ○ Testes unitários e de integração. ● Testes de Integração relacionados: TI1, TI2, TI3, TI4 e TI5. ● Casos de uso relacionados: UC6, UC7, UC8, UC9, UC22, UC25, UC36.
Sprint 4	16/09/2023 – 30/09/2023 (15 dias)	<ul style="list-style-type: none"> ● Implementar <i>backend</i> de métricas de repositório da SupportPlatform-Service: <ul style="list-style-type: none"> ○ Quantidade de arquivos ● Implementar interface de métricas de repositório <ul style="list-style-type: none"> ○ Quantidade de arquivos; ● Implementar <i>backend</i> de filtro de contribuidores das rotas de métricas da SupportPlatform-Service; ● Implementar filtro de contribuidor na página de métricas do repositório; ● Implementar interface de login integrado com GitHub da plataforma; ● Casos de uso relacionados: 	<ul style="list-style-type: none"> ● Implementar <i>backend</i> de <i>issue padronizada</i> no SupportPlatform-Service: <ul style="list-style-type: none"> ○ Criação, atualização, leitura e deleção; ○ Testes unitários e de integração. ● Casos de uso relacionados: UC10, UC39, UC40, UC41 e UC42.

		<p>UC1, UC12 e UC18</p> <ul style="list-style-type: none"> ● Testes de aceitação relacionados: N1T1, N1T2, N1T3 e N6T4. 	
Sprint 5	01/10/2023 – 15/10/2023 (15 dias)	<ul style="list-style-type: none"> ● Implementar interfaces de gerenciar método avaliativo: <ul style="list-style-type: none"> ○ Criação, atualização, leitura e deleção. ● Implementar interface de gerenciar repositórios de um método avaliativo; ● Implementar interface de gerenciar <i>sprints</i> de um método avaliativo: <ul style="list-style-type: none"> ○ Criação, atualização, leitura e deleção. ● Casos de uso relacionadas: UC3, UC4, UC5, UC21, UC33, UC34, UC35, UC37 e UC38; ● Testes de Aceitação relacionados: N3T1, N3T2, N3T3. 	<ul style="list-style-type: none"> ● Implementar <i>cronjob</i> para validar regras de consistência em repositórios no JobScheduler-Service: <ul style="list-style-type: none"> ○ Criação de entrega de regra de consistência; ○ Detecção de <i>squashes</i> e <i>rebases</i>; ○ Integração para abertura de <i>issue</i> padronizada no GitHub. ● Casos de uso relacionados: UC23, UC28, UC29, UC30 e UC31; ● Testes de aceitação relacionados: N7T4, N8T1, e N8T2.
Sprint 6	16/10/2023 – 31/10/2023 (16 dias)	<ul style="list-style-type: none"> ● Implementar interface de gerenciar regras de consistência de um método avaliativo; ● Implementar interface de gerenciar <i>issues</i> padronizadas de um método avaliativo; ● Implementar interface de histórico de <i>commits</i> do repositório; ● Implementar interface de histórico de arquivos do repositório; ● Implementar filtros por <i>sprint</i> na interface de métricas de repositório; ● Implementar interface de configurações do repositório; ● Implementar interface de 	<ul style="list-style-type: none"> ● Implementar serviço de análise estática de repositório com SonarQube: <ul style="list-style-type: none"> ○ Definir banco de dados; ○ Configurar imagem do SonarQube no Docker; ○ Integrar serviços do SonarQube com o SupportPlatform-Service. ● Caso de uso relacionado: UC11.

		<p>entregas de regra de consistência de um repositório;</p> <ul style="list-style-type: none"> ● Implementar <i>backend</i> de sincronizar apenas um repositório da SupportPlatform-Service e da JobScheduler-Service; ● Implementar <i>frontend</i> de sincronizar um repositório; ● Casos de uso relacionados: UC6, UC7, UC9, UC10, UC13, UC14, UC22, UC24, UC25, UC26, UC27, UC31, UC36, UC39, UC40, UC41 e UC42; ● Testes de aceitação relacionados: N2T1, N2T2, N2T4, N2T5, N3T4, N4T1, N4T2, N7T1, N7T2 e N7T3. 	
Sprint 7	01/11/2023 – 15/11/2023 (15 dias)	<ul style="list-style-type: none"> ● Implementar interface de análise estática de código do repositório; ● Implementar interface para duplicar um método avaliativo; ● Correções e aprimoramentos com relação ao <i>feedback</i> da primeira versão estável da plataforma; ● Pagamento de dívidas técnicas não intencionais; ● Execução de testes regressivos da plataforma. ● Casos de uso relacionados: UC11 e UC35; ● Testes de aceitação relacionados: N5T1, N5T2, N5T3, N5T4. 	<ul style="list-style-type: none"> ● Correções e aprimoramentos com relação ao <i>feedback</i> da primeira versão estável da plataforma; ● Pagamento de dívidas técnicas não intencionais; ● Execução de testes regressivos da plataforma.
Sprint 8	16/11/2023 – 26/11/2023 (10 dias)	<ul style="list-style-type: none"> ● Documentar introduções de implantação e utilização; ● Implantar sistema em produção; 	<ul style="list-style-type: none"> ● Documentar introduções de implantação e utilização; ● Implantar sistema em produção;

		<ul style="list-style-type: none"> ● Escrita do Post-mortem do projeto. 	<ul style="list-style-type: none"> ● Escrita do Post-mortem do projeto.
--	--	--	--

Tabela 68. Cronograma de atividades de desenvolvimento da plataforma

8.2 Processo de Implementação

Esta seção aborda detalhadamente o processo de implementação da plataforma projetada, apresentando as etapas e as ferramentas utilizadas para garantir um desenvolvimento eficiente e organizado. Inicialmente, é importante destacar que o desenvolvimento da plataforma segue uma abordagem incremental e iterativa, no qual as entregas são divididas em *sprints* quinzenais. Isso permite que as funcionalidades sejam implementadas progressivamente, possibilitando uma melhor adaptação às necessidades e requisitos do projeto.

Em relação ao controle de versões do código, é adotada a ferramenta Git. Com a adoção dessa ferramenta, é possível rastrear as modificações realizadas, facilitando a colaboração entre os membros da equipe e fornecendo um ambiente seguro para o desenvolvimento. Ademais, todo o código desenvolvido é armazenado em um repositório no GitHub.

Durante o processo de desenvolvimento, são utilizadas *branches* para trabalhar em funcionalidades distintas e, quando uma funcionalidade está concluída, é aberto um *pull request*. Esse mecanismo permite que os membros da equipe revisem as alterações propostas, discutam e ofereçam sugestões, antes de serem incorporadas à *branch* principal. As descrições dos *pull requests* seguem um *template* que permite que o autor especifique, a partir de seleções pré-definidas, mais informações do *pull request*, como qual o tipo de alteração e quais os testes aplicados ao código. Para garantir que todos os *pull requests* seguem o *template*, é utilizada um *workflow* da ferramenta GitHub Actions, que automaticamente adiciona o *template* como descrição do *pull request*.

Ao longo do desenvolvimento das funcionalidades, são utilizadas de estratégias de testes para validar e garantir a qualidade da solução implementada. Além dos testes de aceitação e integração previamente descritos, são realizados testes unitários nos diversos módulos da plataforma. Esses testes são construídos conforme as funcionalidades são implementadas, e desempenham um papel fundamental para identificar erros e problemas, facilitando a manutenção e aprimoramento do *software* ao longo das *sprints*.

A fim de promover a integração e entrega contínua, são aplicadas práticas que visam automatizar o processo de teste e implantação de alterações no código-fonte principal. Por meio da ferramenta GitHub Actions, são definidos fluxos de trabalho personalizados, que realizam testes automatizados e publicam automaticamente as alterações no código-fonte, garantindo a estabilidade e a qualidade do projeto. A Figura 46 apresenta uma lista dos *workflows* do configurados, além dos mais recém-executados.

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with sections like 'Actions' (highlighted), 'New workflow', 'All workflows' (selected), '.github/workflows/auto-pull-request.yaml', 'Compress Images', 'Deploy master branch', 'Deploy Pull Request', 'Jest', 'JobScheduler Service Lint', 'SupportPlatform Service Lint', 'SupportPlatform Web Lint', 'Management', and 'Caches'. The main area is titled 'All workflows' with a subtitle 'Showing runs from all workflows'. It displays '1.167 workflow runs'. The first four runs are listed:

- Merge pull request #329 from ICEI-PUC-Minas-PPLES-TI/fe...** (master) - 5 days ago, 8m 39s
- Feature/help link** - 5 days ago, 1m 36s
- Feature/help link** - 5 days ago, 1m 8s
- Feature/help link** - 5 days ago, 7m 57s

Each run row includes a green checkmark icon, the workflow name, the branch or actor, the time it was run, and a three-dot menu icon.

Figura 46. Workflows de integração e entrega contínua do projeto no GitHub Actions

Para detectar vulnerabilidades de segurança do código, é utilizada Dependabot, uma ferramenta automatizada fornecida pelo GitHub, que nos auxilia na detecção de vulnerabilidades e alerta sobre atualizações de dependências. Essa automação contribui para a manutenção de um ambiente seguro, permitindo que as vulnerabilidades sejam prontamente identificadas e resolvidas. Além disso, para padronização e controle da qualidade do código produzido, todo código submetido em *pull requests* é avaliado por meio de análise estática de código, fazendo comentários quando há algum problema encontrado. Esse processo é realizado de forma autotomizada por meio do GitHub Actions.

Quanto ao controle das tarefas, é utilizado o recurso de Kanban oferecido pelo GitHub Projects, ilustrado na Figura 47. Esse sistema permite registrar as tarefas como “issues”, que podem ser classificadas em colunas representando seu estado atual. As colunas utilizadas são “*To do*” (a fazer), “*In progress*” (em progresso) e “*Done*” (feito). Essa abordagem proporciona uma visão clara do andamento do projeto, permitindo que os membros da equipe movam as tarefas entre as colunas conforme as alterações em seu estado.

Além disso, com o início da implementação da plataforma, foi utilizado um Kanban mais completo, como mostra a Figura 48. Esse Kanban manteve as colunas “*In progress*” e “*Done*” do anterior, possuindo também a coluna “*New*” (novo), para tarefas adicionadas ao longo do desenvolvimento, relacionadas a possíveis implementações futuras. Possui também a coluna “*Backlog*” para itens planejados que ainda não entraram na *sprint*, a coluna “*Ready*” (pronto) para tarefas que já podem ser iniciadas e a coluna “*In Review*”, para tarefas que estão sendo revisadas pelo outro integrante da dupla.

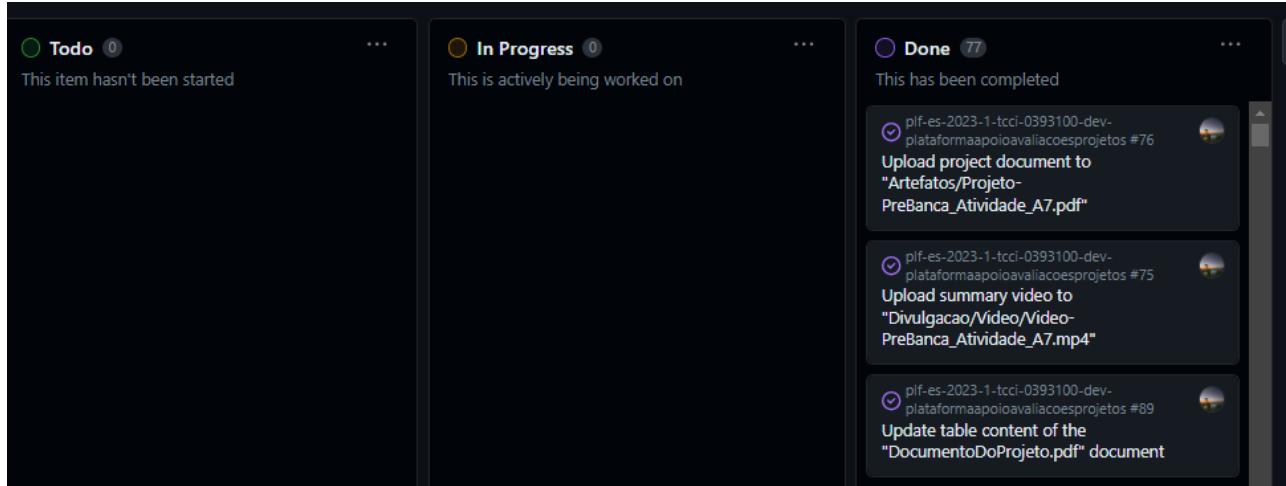


Figura 47. Kanban de divisão de tarefas do projeto

This Kanban board displays the implementation tasks for the project, organized into four columns: 'Ready' (3 items), 'In progress' (1 item), 'In review' (2 items), and 'Done' (110 items). The 'Done' column is the most populated, showing numerous completed tasks. These tasks involve developing interfaces, displaying repository metrics, configuring environments, and implementing specific features like support for platforms and services. Each task is represented by a card with a checkmark icon, a title, and a detailed description.

Figura 48. Kanban de divisão de tarefas de implementação do projeto

9. Post-mortem

Nesta seção, são apresentados relatos das experiências dos desenvolvedores desse sistema. A Seção 9.1 reúne um resumo das impressões positivas observadas. A Seção 9.2 detalha certas vivências desfavoráveis. Na Seção 9.3 são abordadas algumas lições importantes obtidas ao longo do desenvolvimento da aplicação proposta. Por fim, na Seção 9.4 é apresentado o repositório de código remoto do sistema.

9.1 Experiência Positivas

Dentre as experiências positivas destacadas neste projeto, algumas merecem atenção especial pela relevância e impacto. Primeiramente, a utilização do Jest¹¹ em conjunto com o Supertest¹² para a implementação de testes de integração em toda a API foi decisiva. Esta abordagem permitiu alcançar uma cobertura de código de 87.69% do total de 11506 linhas da API, conforme ilustrado na Figura 49. Essa prática revelou-se benéfica ao possibilitar o avanço no desenvolvimento do *backend* independentemente do *frontend*, minimizando falhas e lacunas de funcionalidades.

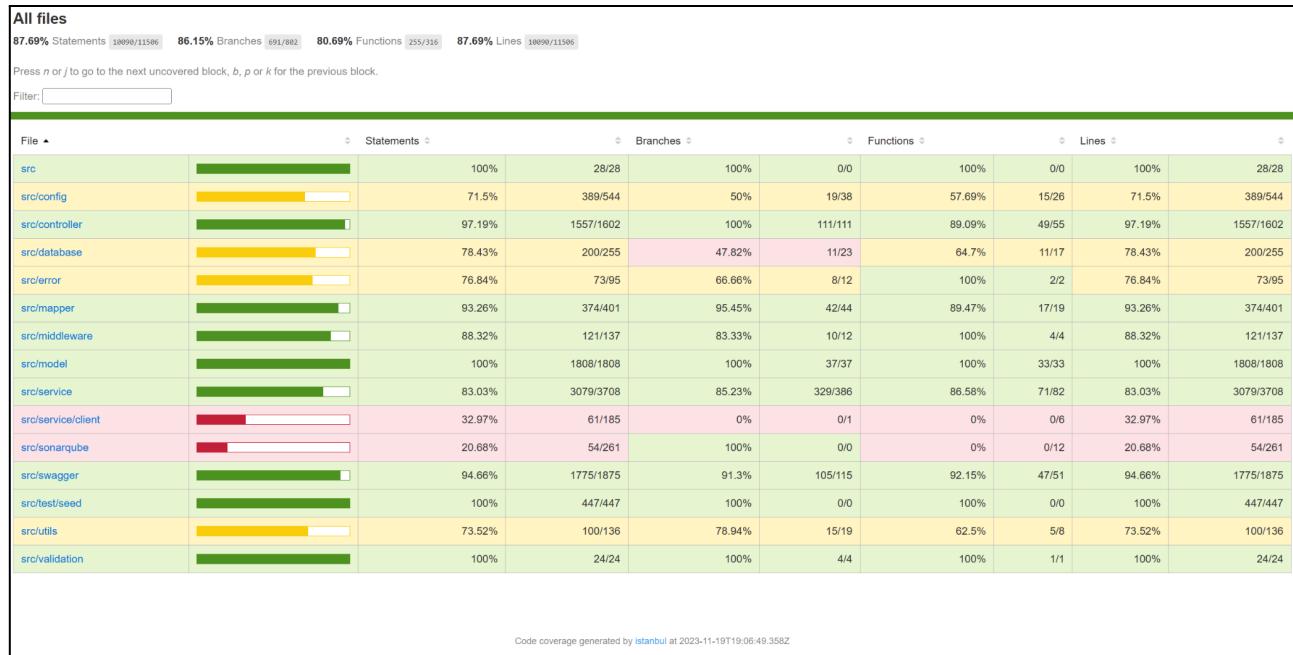


Figura 49. Cobertura de testes de integração da API

Outro aspecto positivo foi a criação de *pipelines* de Integrações Contínuas (CI, do inglês *Continuous Integration*) e Entregas Contínuas (CD, do inglês *Continuous Delivery*), utilizando o GitHub Actions. Foi criada uma *pipeline* de CD especificamente para o *deploy* da *branch master* no ambiente de homologação, apresentada na Figura 50. Essa estratégia facilitou a realização de testes e a utilização do sistema pelo orientador do TCC. Além disso, outra *pipeline* de CD para permitir a implantação imediata de ambientes de desenvolvimento para cada *pull request* aberto, envolvendo mudanças no código, permitiu que ambos os desenvolvedores, além de realizassem revisões de código, poderem também testar o código sem necessitar de rodar localmente as alterações. Também foi desenvolvida uma *pipeline* de CI para assegurar que todos os testes e *lints* dos serviços *backend* e *frontend* fossem executados a cada novo *pull request*, garantindo um código limpo e com menor chance de falhas a cada alteração.

¹¹ <https://jestjs.io/>

¹² <https://github.com/ladjs/supertest>

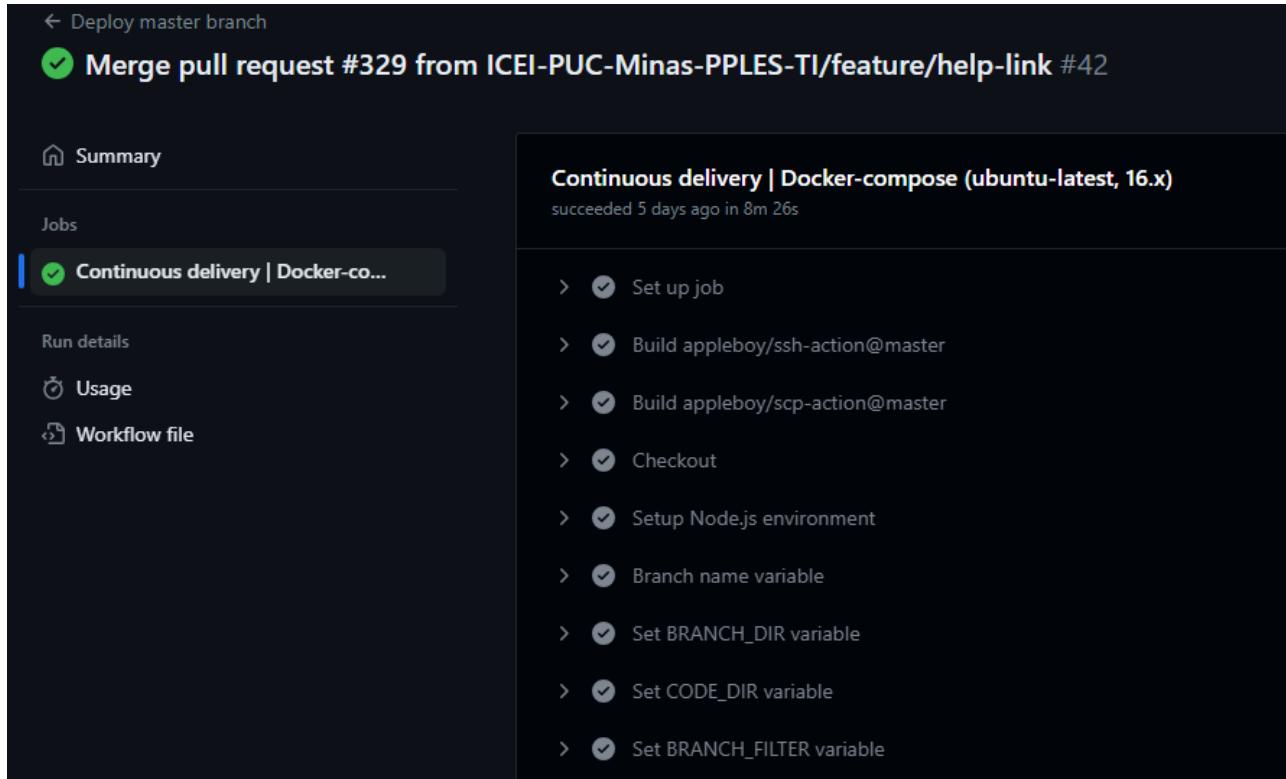


Figura 50. Workflow de CD de *deploy* da *branch* principal do GitHub Actions

Além desses, outro aspecto positivo foram as reuniões de apresentação da plataforma desenvolvida para o cliente. Essas reuniões consistiam realizar demonstrações abrangentes de todas as funcionalidades da plataforma. Assim, permitiu-se ao cliente realizar a validação dos testes de aceitação, culminando na aprovação da plataforma, e garantindo confiança no produto que desenvolvido.

Por último, a implementação de um sistema Kanban utilizando o GitHub Projects, combinado com a metodologia de *sprints*, foi fundamental para do desenvolvimento ágil do projeto. Esta abordagem permitiu uma gestão eficiente de tarefas, controle de tempo e demandas, assegurando que o projeto progredisse de forma organizada e eficaz.

9.2 Experiência Negativas

No processo de desenvolvimento da plataforma, algumas experiências negativas foram particularmente desafiadoras. A primeira e mais notável foi a complexidade extremamente alta na sincronização de dados entre os repositórios do GitHub, que funcionam com um banco de dados em grafos contendo *branches*, *commits*, e arquivos alterados por *commits*. Foi necessário capturar e salvar esses dados em um banco de dados relacional (MySQL) da plataforma. Esse aspecto tornou as buscas, relacionamentos, registros e consultas dos dados sincronizados extremamente complexos.

Outro grande desafio enfrentado foi a complexidade ainda maior no cálculo de métricas dos dados dos repositórios, que incluem *branches*, *commits*, arquivos, e contribuidores de cada *commit* e arquivo. Além disso, para a criação de gráficos foi necessário relacionar os dados sincronizados do

GitHub com períodos temporais delimitados pelas datas de início de fim de *sprints* e além de filtro por contribuidores. Esse processo acabou demandando mais tempo do que o planejado para o desenvolvimento dos serviços de consultas ao MySQL no *backend* e a montagem dos gráficos no *frontend*.

Por fim, a integração com o SonarQube apresentou-se como uma experiência negativa particularmente árdua. A tentativa de executar análises estáticas de código de forma genérica para quaisquer repositórios da organização do GitHub contrariou o funcionamento típico do SonarQube, que requer configurações específicas para cada repositório, linguagem e *framework*. Além disso, a biblioteca do SonarScanner utilizada para a integração com o SonarQube apresentou várias lacunas funcionais, ausência de tipagens e tratamentos de erros, tornando a integração uma tarefa extremamente desafiadora e frustrante. Apesar disso, foi possível implementar a funcionalidade de análise estática do código, a partir de uma configuração genérica, que serve para um amplo conjunto de linguagens.

9.3 Lições Aprendidas

Ao refletir sobre as lições aprendidas durante o desenvolvimento deste projeto, percebe-se a importância de um planejamento detalhado e a necessidade de adaptabilidade. Uma das lições mais significativas foi a descoberta de que algumas funcionalidades nativas presumidas do GitHub, como a identificação de *force pushes* nos repositórios, na realidade não existiam. Isso ressalta a importância de uma compreensão aprofundada das limitações das ferramentas utilizadas antes de incluí-las no escopo do projeto. Sendo válido ressaltar que havia sido mapeado uma necessidade de detectar *branches* inativas que não passaram por *merge* até o fim de uma *sprint* e criar *issues* como forma de alerta aos alunos que esta não é uma boa prática. No entanto, essa funcionalidade foi posteriormente removida dos casos de uso. A decisão baseou-se no entendimento de que uma *branch* pode pertencer a uma *sprint* futura ou estar sendo utilizada apenas para testes. Além disso, considerou-se que o GitHub não fornece informações diretas sobre a realização de *merges*.

Outro aprendizado vital foi o valor da integração contínua para testes. Essa prática provou ser essencial, salvando o projeto de várias ocasiões em que *pull requests* poderiam ter quebrado funcionalidades existentes. Além disso, a utilização do Kanban do GitHub Projects, em conjunto com *pull requests*, *milestones* e *issues*, foi extremamente eficaz na organização e no acompanhamento das entregas.

A decisão de dedicar uma *sprint* final de desenvolvimento exclusivamente para o pagamento de dívidas técnicas foi essencial, permitindo resolver pontos técnicos e funcionalidades incompletas que haviam sido adiadas. Ficou evidente também que integrações com sistemas externos são mais complexas e problemáticas do que inicialmente previsto, exigindo um tempo de projeto mais extenso.

Quanto à geração de métricas e gráficos, constatou-se que sua complexidade pode escalar significativamente com o tamanho e a complexidade do banco de dados. A experiência prévia dos membros da equipe nas tecnologias escolhidas foi um fator determinante para a eficiência das entregas.

Por fim, a definição clara do escopo, das funcionalidades, a modelagem de diagramas no Astah e a criação de protótipos no Figma foram fundamentais para a entrega de um software

completo e robusto. Essa abordagem trouxe uma segurança e a integridade na entrega do produto final.

9.4 Repositório do Trabalho

O código-fonte deste projeto, juntamente com todos os artefatos gerados ao longo do desenvolvimento deste trabalho, estão disponíveis para consulta e uso no repositório do GitHub. Este repositório inclui não apenas o código em si, mas também documentação detalhada, registros de alterações, e demais materiais relevantes criados durante as fases de planejamento, desenvolvimento e implementação do sistema. Acesse o repositório pelo link a seguir: <https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2023-1-tcci-0393100-dev-plataformaapoiavaliacoesprojetos>