UNIVERSIDADE DO ESTADO DO AMAZONAS

Um juiz on-line educacional para disciplinas de introdução à programação da EST/UEA

Carlos Alberto Hagge da Cunha Filho

Manaus
Dezembro 2024

UNIVERSIDADE DO ESTADO DO AMAZONAS

Carlos Alberto Hagge da Cunha Filho

Um juiz on-line educacional para disciplinas de introdução à programação da EST/UEA

Trabalho de Conclusão de Curso apresentado à banca avaliadora do Curso de Engenharia de Computação, da Escola Superior de Tecnologia, da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenheiro de Computação.

Orientador:
Prof. MSc. Flávio José Mendes Coelho

Manaus
Dezembo 2024

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).
Sistema Integrado de Bibliotecas da Universidade do Estado do Amazonas.

C972j Cunha Filho, Carlos Alberto Hagge da Cunha Filho
Um juiz on-line educacional para disciplinas de introdução à programação da EST/UEA / Carlos Alberto Hagge da Cunha Filho Cunha Filho . Manaus : [s.n], 2024.
34 f.: color.; 21,0 cm.

TCC - Graduação em Engenharia de Computação- Universidade do Estado do Amazonas, Manaus, 2024.
Inclui Bibliografia.
Inclui Apêndice.
Orientador: Flávio José Mendes Coelho.

1. Juiz on-line educacional. 2. Learning Analytics. 3. ICONIX. 4. IDE web. I. Flávio José Mendes Coelho (Orient.) II. Universidade do Estado do Amazonas. III. Título

CDU(1997)62:004

Universidade do Estado do Amazonas - UEA
Escola Superior de Tecnologia - EST
Curso de Engenharia de Computação

FOLHA DE APROVAÇÃO

Um Juiz On-line Educacional para Disciplinas de Introdução à Programação da EST/UEA

Carlos Alberto Hagge da Cunha Filho

Trabalho de Conclusão de Curso defendido e aprovado pela banca avaliadora constituída pelos professores:

Prof. MSc. Flávio José Mendes Coelho
Presidente

Prof. Dr. Luis Cuevas Rodriguez
Avaliador(a)

Profa. Dra. Marcela Sávia Picanco Pessoa
Avaliador(a)

Manaus, 13 de Dezembro de 2024.

Agradecimentos

Primeiramente, agradeço a meus amados pais, Ana Cristina e Carlos, que desde sempre estiveram ao meu lado, com todo o apoio possível, e sempre oferecendo a mim as melhores oportunidades.

Com imensa gratidão gostaria de agradecer também meu orientador, Flávio José Mendes Coelho. Sua orientação foi fundamental para a elaboração deste Trabalho de Conclusão de Curso. Contei sempre com seu apoio, paciência e conhecimentos durante meu percurso.

Finalmente, agradeço a todos que de alguma forma colaboraram durante minha jornada.

Resumo

Este Trabalho de Conclusão de Curso apresenta o projeto e implementação de um juiz on-line de código, desenvolvido como ferramenta pedagógica para auxiliar professores e alunos nas disciplinas introdutórias de programação da Escola Superior de Tecnologia da Universidade do Estado do Amazonas (EST/UEA). O sistema inclui um ambiente web de edição de código que captura as interações dos usuários com o sistema em nível de pressionamento de teclas, e um banco de dados para armazenar esses registros. Como resultado, foram desenvolvidos todos os artefatos de software necessários, guiados por um processo de software ágil, com uma arquitetura específica para juízes on-line educacionais. Testes de aceitação foram realizados para validar o sistema, e trabalhos futuros incluem melhorias na usabilidade, testes com turmas reais e aplicação de *learning analytics*.

Palavras chave: Juiz on-line educacional; Learning Analytics; ICONIX; IDE web.

Abstract

This works presents the design and implementation of an online code judge system, developed as an educational tool to assist teachers and students in introductory programming courses at the School of Technology of the Amazonas State University (EST/UEA) and to support future research in Learning Analytics. The system includes a web-based code editor that captures user interactions and a database to store these logs. As a result, all necessary software artifacts were developed, guided by an agile software process and a specific architecture for educational online judges. Acceptance tests were conducted to validate the system, and future work includes improving usability, testing with real classes, and analyzing the captured data.

Keywords: Online code judge; Learning Analytics; ICONIX; IDE web.

Conteúdo

1 Introdução **8**
1.1 Justificativa . . . 9
1.2 Objetivos gerais e específicos . . . 9
1.3 Metodologia . . . 9
1.3.1 O processo de desenvolvimento de software ICONIX . . . 10
1.3.2 Tecnologias utilizadas . . . 12
1.4 Organização do trabalho . . . 13

2 Fundamentação teórica **14**
2.1 Sistemas de correção automática de código . . . 14
2.2 Sistemas juízes on-line . . . 14
2.3 Detecção de plágio . . . 15
2.4 *Learning Analytics* . . . 16
2.5 Considerações sobre a segurança . . . 16

3 Trabalhos relacionados **18**

4 Resultados **21**
4.1 Análise de requisitos . . . 21
4.1.1 Requisitos funcionais . . . 21
4.1.2 Requisitos não funcionais . . . 22
4.2 Análise e projeto preliminar . . . 22
4.2.1 Caso de uso CDU-1: Aluno ingressa em uma turma utilizando o código da turma . . . 22
4.2.2 Caso de uso CDU-2: Aluno envia a solução de um problema . . . 23
4.2.3 Caso de uso CDU-3: Aluno vê os resultados das suas soluções . . . 23
4.2.4 Análise de robustez . . . 23
4.3 Projeto detalhado . . . 24
4.4 Implementação . . . 26
4.4.1 Arquitetura do sistema . . . 26
4.4.2 Amostra da interface do sistema . . . 26

5 Considerações finais **33**
5.1 Trabalhos futuros . . . 33

APÊNDICES **38**

A Diagramas de robustez **39**

B Diagramas de sequência **43**

Lista de Figuras

1.1 Diagrama do processo de desenvolvimento de software ICONIX. Fonte: (ROSENBERG, 2001) . . . 11

3.1 Arquitetura de um juiz on-line. Fonte: (WATANOBE et al., 2022) . . . 19
3.2 Exemplo de dados coletados pelo CodeBench, onde o usuário navega e digita `print`. . . . 20

4.1 Diagrama de domínio. . . . 21
4.2 Diagrama de caso de uso, apresentando um subconjunto de funcionalidades. . . . 23
4.3 Diagrama de robustez para RF-007 . . . 24
4.4 Diagrama de robustez para RF-009 . . . 24
4.5 Diagrama de robustez para RF-010 . . . 24
4.6 Diagrama de sequência RF-007 . . . 25
4.7 Diagrama de sequência RF-009 . . . 26
4.8 Diagrama de sequência RF-010 . . . 27
4.9 Diagrama de classe. . . . 28
4.10 Diagrama de arquitetura . . . 29
4.11 Página de um problema, mostrando seu enunciado. . . . 29
4.12 Página de um problema, mostrando entrada, saída, arquivos anexados e datas. 30
4.13 Página do editor, mostrando dentro dele o código e funcionalidades de coloração de sintaxe, documentação e análise de símbolos. . . . 30
4.14 Área de submissões da página de um problema. . . . 31
4.15 Página de administração mostrando lista de eventos de edição armazenados no banco de dados. . . . 31
4.16 Página mostrando detalhes de um único evento de edição. . . . 32

A.1 Diagrama de robustez para RF-001 . . . 39
A.2 Diagrama de robustez para RF-002 . . . 39
A.3 Diagrama de robustez para RF-003 . . . 40
A.4 Diagrama de robustez para RF-004 . . . 40
A.5 Diagrama de robustez para RF-005 . . . 40
A.6 Diagrama de robustez para RF-006 . . . 41
A.7 Diagrama de robustez para RF-007 . . . 41
A.8 Diagrama de robustez para RF-008 . . . 41
A.9 Diagrama de robustez para RF-009 . . . 41
A.10 Diagrama de robustez para RF-010 . . . 42
A.11 Diagrama de robustez para RF-011 . . . 42
A.12 Diagrama de robustez para RF-012 . . . 42

B.1 Diagrama de sequência RF-001 . . . 43
B.2 Diagrama de sequência RF-002 . . . 44
B.3 Diagrama de sequência RF-003 . . . 44

B.4 Diagrama de sequência RF-004 45
B.5 Diagrama de sequência RF-005 45
B.6 Diagrama de sequência RF-006 46
B.7 Diagrama de sequência RF-007 46
B.8 Diagrama de sequência RF-008 46
B.9 Diagrama de sequência RF-009 47
B.10 Diagrama de sequência RF-010 47
B.11 Diagrama de sequência RF-011 48
B.12 Diagrama de sequência RF-012 48

Lista de Tabelas

2.1 Alguns exemplos de juízes on-line. 15

Capítulo 1

Introdução

Atualmente, o ensino de programação é cada vez mais importante e presente em uma quantidade crescente de cursos da área de exatas. A alta quantidade de alunos estudando programação acaba gerando sobrecarga em professores, pela quantidade de código a ser corrigido manualmente. Esta demanda grande e crescente tem motivado o aumento da adoção de sistemas de correção automática de código. Outro fator motivador é o uso de ensino à distância (EaD) e híbrido por muitas instituições após a pandemia de COVID-19, gerando a necessidade de ferramentas para avaliar tarefas de programação de forma remota (COMBÉFIS, 2022).

Aprender programação pode ser uma tarefa difícil, principalmente para alunos que não são da área de computação, mas que precisam aprender a programar para fazer alguma disciplina de seus cursos. Portanto, ferramentas que auxiliem neste aprendizado são bem-vindas. Sistemas de correção automática de código, pela rapidez com a qual podem produzir *feedbacks* aos alunos, pode ser um fator positivo no processo de aprendizado de programação (RIBEIRO et al., 2018).

Sistemas de correção automática de código para trabalhos e avaliações práticas de disciplinas de programação têm sido desenvolvidos desde a década de 1960 (FORSYTHE; WIRTH, 1965) e têm ajudado tanto professores a reduzir sua carga de esforço na correção manual desses trabalhos e avaliações, como também proporcionam aos alunos uma experiência de aprendizado mais dinâmica e prática. Ferramentas que oferecem esse tipo de automação são chamadas de ferramentas de avaliação automática de código (ALA-MUTKA, 2005). Um subtipo dessas ferramentas é o juiz on-line, uma ferramenta de avaliação automática de código com características específicas. Dentre essas características, destaca-se o requisito de se avaliar dinamicamente programas submetidos ao sistema por usuários (WASIK et al., 2018). Juízes on-line estão se tornando populares em contextos como competições, recrutamento e educação. Alguns exemplos de juízes on-line de código incluem o Beecrowd (anteriormente, URI Online Judge), Codeforces, HackerRank (WASIK et al., 2018) e CodeBench (CARVALHO; OLIVEIRA; GADELHA, 2016). Desses, o Beecrowd, Codeforces e HackerRank possuem versões públicas acessíveis através da internet.

As funcionalidades padrão esperadas em um juiz on-line orientado a competições são a execução e customização de casos de teste[1], o compartilhamento de um banco de problemas para competições e treinamento, e um *ranking* de competidores por competição (WASIK et al., 2018). Por outro lado, as funcionalidades desejáveis em um juiz on-line educacional incluem:

- A disponibilização de um editor de código na interface *web* do juiz on-line com funcionalidades comparáveis àquelas encontradas em editores de código tradicionais para *desktop*

[1] Testes de caixa preta em que o programa a ser testado recebe um conjunto de entradas e precisa gerar um conjunto de saídas igual a um conjunto de saídas esperadas, previamente definidas.

tais como Visual Studio Code[2], PyCharm[3] e Eclipse[4];

- A coleta de dados em tempo real das sessões de codificação dos usuários (texto digitado, entrada e saída do foco na página, cópia e colagem de texto, etc), para posterior análise de dados;
- Gerenciamento de listas de exercícios e avaliações, por disciplina e professor, com armazenamento e compartilhamento desses componentes.

1.1 Justificativa

Os juízes on-line atualmente disponíveis na internet ou de código aberto não são gratuitos ou não atendem completamente às particularidades das disciplinas de programação da EST/UEA. Há juízes on-line educacionais que implementam as funcionalidades já mencionadas tais como o CodeBench (CARVALHO; OLIVEIRA; GADELHA, 2016) e o Codio[5]. Porém, o desenvolvimento de uma ferramenta ajustada às particularidades da EST/UEA, facultaria a criação de novas funcionalidades e o ajuste daquelas existentes, sob demanda. Além disso, uma ferramenta local serviria como fonte para pesquisas em *Learning Analytics* na área de Educação em Computação, assim como poderá servir de plataforma para o desenvolvimento de futuros trabalhos de conclusão de curso, beneficiando os alunos dos cursos de computação da EST/UEA.

1.2 Objetivos gerais e específicos

Este trabalho de conclusão de curso tem como objetivo apresentar o projeto e a implementação de um juiz on-line educacional para apoiar as atividades pedagógicas de professores e alunos das disciplinas de introdução à programação da Escola Superior de Tecnologia (EST) da UEA.

(a) Projetar e implementar um sistema on-line de execução de casos de teste para avaliar códigos enviados por alunos;

(b) Integrar ao sistema um editor on-line de código com recursos necessários para a codificação de programas em C++ e Python, com funcionalidades usuais de IDEs;

(c) Criar uma interface de usuário (IU) on-line para o envio de código através de um navegador;

(d) Prover um ambiente seguro (chamado de *sandbox*) para a execução dos códigos dos alunos;

(e) Coletar as interações dos alunos com o editor (em nível de digitação) e enviá-las para o servidor, para a construção de um *dataset*.

1.3 Metodologia

Para este trabalho, a metodologia se dividiu em quatro partes principais:

1. **Levantamento bibliográfico:** sobre juízes on-line e as tecnologias utilizadas para se desenvolver um juiz on-line.
2. **Seleção das tecnologias:** a partir das pesquisas realizadas, foram selecionadas as tecnologias para a implementação do projeto: Python (linguagem de programação), Django (*framework* web), Monaco (componente de editor de código fonte), entre outras.

[2]https://code.visualstudio.com/
[3]https://www.jetbrains.com/pycharm/
[4]https://eclipseide.org/
[5]https://www.codio.com/

3. **Seleção do processo de desenvolvimento de software:** diferentes processos foram avaliados e adotou-se o processo de desenvolvimento de software ICONIX, por ser um processo ágil e simples (ROSENBERG, 2001).

4. **Projeto e desenvolvimento do software:** as etapas do ICONIX foram seguidas: análise de requisitos, análise e projeto preliminar, projeto, implementação. Os artefatos de cada etapa foram gerados (diagramas de caso de uso, domínio, robustez e sequência), e ao final da implementação foram executados os testes de aceitação.

5. **Testes de aceitação:** Para validar a ferramenta desenvolvida, foram realizados testes de aceitação, executando suas funcionalidades e verificando se estão corretamente implementadas.

1.3.1 O processo de desenvolvimento de software ICONIX

Como citado anteriormente, o ICONIX (ROSENBERG, 2001) foi utilizado como processo de desenvolvimento de software para o sistema apresentado neste trabalho. É um processo ágil, orientado a caso de uso e que preza por uma abordagem simplificada. Se comparado ao RUP (*Rational Unified Process* ou Processo Unificado da Rational) (KRUCHTEN, 2004) e às práticas de XP (*Extreme Programming*) (BECK; ANDRES, 2004), o ICONIX é mais simples que o RUP e mais complexo que XP (ROSENBERG, 2007).

O ICONIX possui procedimentos que visam partir da etapa da definição dos casos de uso até a etapa da escrita do código de forma rápida, eficiente e flexível. Três características chave do processo são: o uso simplificado da UML (do inglês: *Unified Modeling Language*) (BOOCH; RUMBAUGH; JACOBSON, 1999), o alto grau de rastreabilidade do processo e a execução iterativa e incremental.

Conforme ilustra a Figura 1.1, o processo agrupa os artefatos a serem produzidos em dois grandes modelos: o modelo dinâmico inclui os artefatos que modelam comportamentos (casos de uso, diagramas de robustez e diagramas de sequência UML); o modelo estático inclui os artefatos que modelam a parte estrutural do sistema (modelo de domínio e diagrama de classes UML).

Quanto à execução, o processo atravessa quatro etapas:

1. **Análise de requisitos:** os requisitos são levantados e uma prototipação rápida do sistema proposto é desenvolvida. A partir desses artefatos, os objetos de domínio e seus relacionamentos de generalização e agregação são identificados e expressos em um modelo de domínio (um diagrama de classes de alto nível). Os requisitos funcionais são alocados em seus casos de uso correspondentes.

2. **Análise e projeto preliminar:** as descrições dos casos de uso são escritas com seus caminhos base e alternativos. A análise de robustez é feita, visando identificar e modelar as interações entre os objetos do sistema de forma clara e concisa, estabelecendo as responsabilidades de cada objeto. Para cada caso de uso, os objetos que participam do cenário escolhido são identificados, e o diagrama de classes/domínio são atualizados com novos objetos e novos atributos de objetos na medida que são descobertos.

3. **Projeto:** o comportamento do sistema é descrito, detalhando as mensagens que devem ser passadas entre objetos e seus métodos. Os diagramas de sequência são criados e o diagrama de classes é atualizado. O modelo estático é finalizado com a adição de informações detalhadas do projeto. É verificado se o projeto satisfaz os requisitos identificados.

4. **Implementação:** o código é escrito e/ou gerado, são executados testes unitários, de integração e aceitação, e os diagramas são atualizados se necessário.

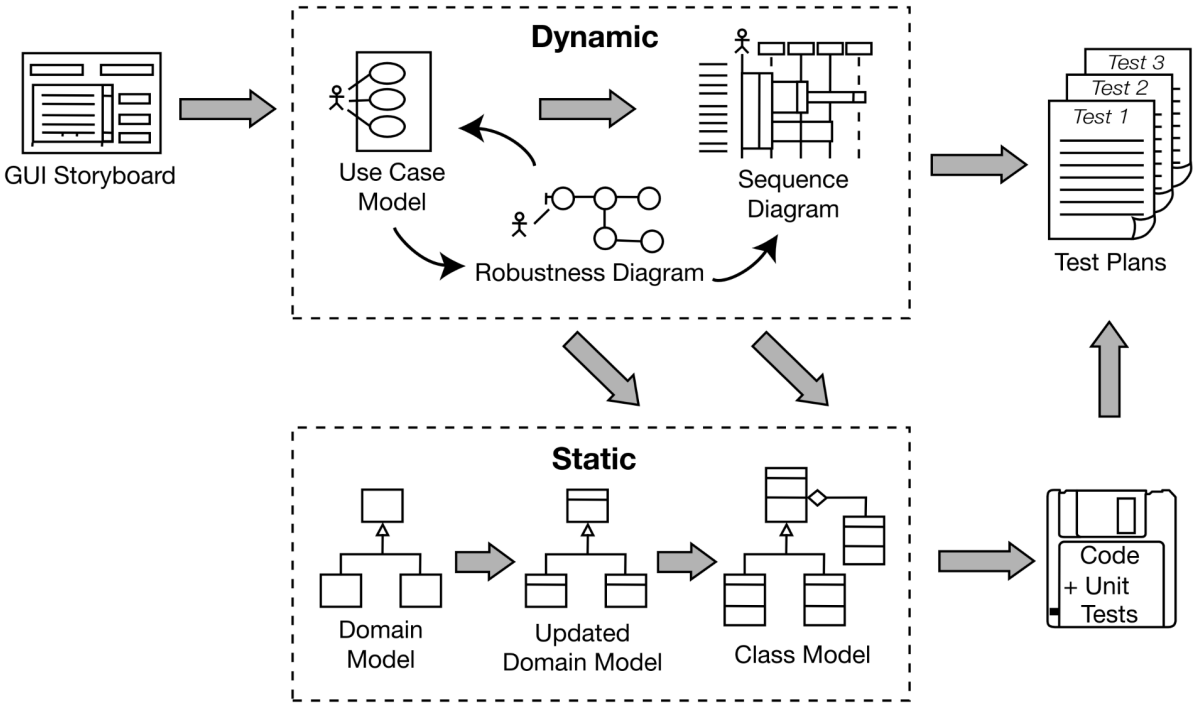

Figura 1.1: Diagrama do processo de desenvolvimento de software ICONIX. Fonte: (ROSENBERG, 2001)

A análise de robustez mencionada anteriormente é uma etapa essencial no processo ICONIX, pois estabelece a ligação entre a descrição textual de um caso de uso e seu diagrama de sequência correspondente. Esse processo consiste em interpretar as sentenças do texto do caso de uso e, para cada uma delas, identificar e desenhar os elementos envolvidos: ator(es), objeto(s) de interface, objeto(s) de entidade e objeto(s) de controle. Em seguida, as conexões entre os objetos são definidas, respeitando quatro regras (ROSENBERG, 2001):

(a) Atores se comunicam apenas com objetos de interface;

(b) Objetos de interface se comunicam com objetos de controle e atores;

(c) Objetos de entidade se comunicam apenas com objetos de controle;

(d) Objetos de controle se comunicam com todos os outros objetos, mas não se comunicam com atores.

Durante o período de TCC I, as duas primeiras etapas do ICONIX foram realizadas: **análise de requisitos** e **análise e projeto preliminar**. Um protótipo foi implementado com as funcionalidades julgadas mais importantes, que seriam o editor *on-line* com coleta de digitações do editor na IDE, envio de códigos através da IU, e execução dos casos de testes nos programas enviados. Ao final do desenvolvimento do protótipo, os testes de suas funcionalidades foram realizados.

Durante o período de TCC II, o mesmo processo foi seguido, porém executando todas as etapas do ICONIX. Foi feita outra iteração do sistema, contendo todas as funcionalidades modeladas. Novamente foram realizados testes no sistema para validar seu funcionamento.

1.3.2 Tecnologias utilizadas

Framework web

A linguagem de programação escolhida para a implementação do *backend* foi Python, na sua versão 3.10.1, que oferece o pacote Django[6], essencial para o desenvolvimento deste trabalho.

Django é um *framework* de desenvolvimento *web* para Python [7]. Utiliza o padrão de *design Model-View-Template* (MVT), baseado no *Model-View-Controller* (MVC) (Django Software Foundation, 2005b) e é empregado principalmente para implementar sistemas de arquitetura monolítica (Django Software Foundation, 2005a), um modelo mais clássico com três camadas: (a) código de interface; (b) lógica de negócio; (c) banco de dados. Aplicações monolíticas são executadas como um único processo do ponto de vista do sistema operacional (BLINOWSKI; OJDOWSKA; PRZYBYŁEK, 2022).

O *framework* é focado em oferecer uma experiência de desenvolvimento rápida, com uma variedade de recursos que visam agilizar as tarefas de desenvolvimento: telas de administração de banco de dados, mapeamento objeto-relacional (ORM, do inglês: *Object Relational Mapper*), criação automatizada de *scripts* para a migração do banco de dados, e um sistema de autenticação e autorização de usuários (Django Software Foundation, 2005a).

Editor de código fonte

O componente de editor de código utilizado no sistema dessa proposta é o Monaco (Microsoft Corporation, 2013). Monaco é um componente de editor de código fonte baseado em tecnologias *web*, com sua base de código extraída do componente de edição utilizado no Visual Studio Code (Microsoft Corporation, 2013).

Outro componente de editor *web* inicialmente avaliado para uso neste trabalho foi o CodeMirror, que foi descartado devido ao suporte inferior à linguagem C++ e ao *Language Server Protocol* (LSP), quando comparado com o Monaco, tornando a experiência do usuário mais limitada devido à falta de autocompletar e referências a símbolos (HAVERBEKE, 2007).

Suporte a linguagens no editor

O *Language Server Protocol* (LSP) define um protocolo baseado na troca de mensagens no formato *JavaScript Object Notation-Remote Procedure Call* (JSON-RPC) (JSON-RPC Working Group, 2013), entre um editor de código ou ambiente integrado de desenvolvimento (em inglês: *Integrated Development Environment* IDE) e um servidor de linguagem, que fornece funcionalidades como: autocompletar, ir para definição de um símbolo, procurar referências, entre outras.

O objetivo do LSP é padronizar a comunicação entre editores de código e *softwares* de desenvolvimento que fornecem as funcionalidades citadas anteriormente. Dessa forma, editores de código podem ser desacoplados das outras ferramentas de desenvolvimento, agindo apenas como um terminal que exibe o texto do código e se comunica com um ou mais servidores LSP. Esses servidores se encarregam de analisar o código atual, do projeto e de bibliotecas instaladas. Portanto, um editor de código que implementa o LSP não necessita de informação além do que é mostrado na tela (Microsoft Corporation, 2016). Geralmente, um editor utiliza diferentes servidores LSP para oferecer suporte a diferentes linguagens. A maioria das linguagens possui multiplas implementações de servidores, cada um com um conjunto de funcionalidades diferente (Microsoft Corporation, 2017; Sourcegraph Inc., 2016). Tais funcionalidades oferecidas pelo LSP dão suporte ao desenvolvimento de ferramentas de desenvolvimento remotas,

[6] https://www.djangoproject.com/
[7] https://www.python.org/

como em uma interface web (Microsoft Corporation, 2016). No *frontend* do projeto apresentado neste trabalho é usado exatamente para tal finalidade, em seu editor de código baseado em Monaco (Microsoft Corporation, 2013)

Para permitir o uso do LSP com a instância do Monaco presente na página do editor *web*, é necessário estabelecer uma conexão *WebSocket* (MELNIKOV; FETTE, 2011) entre o componente executando no navegador do cliente e um servidor LSP.

Compiladores e interpretadores

Inicialmente será implementado o suporte para a linguagem C++, uma linguagem compilada, e o compilador escolhido foi o g++, fornecido pelo GNU Compiler Collection (GCC), um conjunto de compiladores desenvolvido pelo Projeto GNU (Free Software Foundation, 1987). O g++ é utilizado ao avaliar as soluções, compilando os códigos enviados pelos usuários, antes da execução dos casos de teste no arquivo executável gerado.

Futuramente, será implementado o suporte para as linguagens C e Python para auxiliar diversas disciplinas de programação da EST/UEA. Para a linguagem C, o GCC fornece o compilador gcc, e para Python, que é interpretado, o código será executado através de uma ou mais versões do interpretador do Python.

1.4 Organização do trabalho

Este trabalho de conclusão de curso é dividido da seguinte maneira:

- **Capítulo 1 - Introdução:** introduz o tema deste trabalho, a justificativa para sua elaboração, os seus objetivos e a metodologia utilizada no trabalho.
- **Capítulo 2 - Fundamentação teórica:** apresenta e detalha conceitos fundamentais utilizados na elaboração deste trabalho, como sistemas de correção automática de código, juízes on-line e *Learning Analytics*.
- **Capítulo 3 - Trabalhos relacionados:** apresenta trabalhos utilizados durante a revisão da literatura. Os trabalhos levantados nesta seção possuem similaridades com este Trabalho de Conclusão de Curso e serviram como referência para a implementação de elementos do sistema desenvolvido.
- **Capítulo 4 - Resultados:** exibe os resultados alcançados durante a realização deste trabalho, incluindo os artefatos gerados durante o processo de desenvolvimento e os resultados dos testes do sistema desenvolvido.
- **Capítulo 5 - Considerações finais:** discute os resultados alcançados, as limitações do projeto e considerações para trabalhos futuros.

Capítulo 2

Fundamentação teórica

2.1 Sistemas de correção automática de código

Um sistema de correção automática de código tem como objetivo a análise automatizada de códigos submetidos a ele (ALA-MUTKA, 2005). Sistemas de correção automática de código são utilizados, por exemplo, em cursos de programação para avaliar resoluções de estudantes, geralmente executando o programa a ser avaliado e comparando as suas saídas com saídas esperadas. O método de avaliação citado no exemplo é chamado de dinâmico, porém os sistemas de correção automática de código não se limitam a apenas esse método de avaliação. Também existe a avaliação estática, as principais características dos dois métodos são (ALA-MUTKA, 2005):

1. **Avaliação dinâmica:** o sistema executa o programa a ser avaliado, uma ou mais vezes. Da execução do programa são derivadas as métricas que serão analisadas, por exemplo, se as saídas para cada entrada estão de acordo com os requisitos especificados, ou se o programa possui a eficiência desejada em termos de tempo de processamento ou uso de memória durante a execução.

2. **Avaliação estática:** o sistema analisa diretamente o código do programa, sem executá-lo. A maioria dos métodos de análise estática depende do código do programa ser sintaticamente e semanticamente correto. Algumas métricas que podem ser analisadas são o estilo do código, condições impossíveis (partes do código que nunca são executadas) e sua complexidade ciclomática (MCCABE, 1976). Um caso de uso especial para análise estática é a detecção de plágio. Diversas ferramentas incluem alguma forma de comparação de códigos-fonte para detectar eventuais casos de plágio entre usuários.

No sistema apresentado neste trabalho, o método de avaliação utilizado é o dinâmico, pois a maior parte dos sistemas de correção automática de código educacionais utiliza tais avaliações (ALA-MUTKA, 2005), e buscou-se seguir esse padrão pela sua familiaridade para os professores e alunos.

2.2 Sistemas juízes on-line

Sistemas juízes on-line, ou apenas, juízes on-line, são uma subcategoria de sistema de correção automática de código, se caracterizando pela avaliação confiável de programas enviados através da internet. Esses sistemas são amplamente utilizados em avaliações de cursos de programação e em processos de recrutamento. Neles, os programas são compilados, quando necessário, e

testados em um ambiente homogêneo[1] (WASIK et al., 2018).

Juízes on-line, geralmente, efetuam o procedimento de avaliação de soluções em três passos:

1. **Fase de envio:** durante essa fase, o código enviado, se necessário, é compilado e é verificado se pode ser executado no ambiente de avaliação. Em caso positivo, prossegue-se para a fase de análise.
2. **Fase de análise:** nesta fase a infraestrutura do servidor executa o código, aplicando um conjunto de casos de teste específicos para um problema. Para cada caso de teste, o código enviado é executado e o sistema verifica se: (1) durante o processo de execução não ocorreram erros; (2) nenhuma limitação de recursos foi excedida; e (3) as saídas do processo durante o caso de teste correspondem às saídas esperadas na definição do problema.
3. **Fase de pontuação:** na pontuação, o placar associado à solução enviada é computado baseado nos resultados dos casos de teste executados na fase anterior.

Um componente essencial de um juiz on-line é o motor ou mecanismo de avaliação (*evaluation engine*). Esse componente analisa as soluções enviadas pelos usuários, usualmente seguindo os três passos descritos anteriormente. O primeiro passo normalmente é simples e depende da execução de um compilador compatível com o código enviado. Os passos subsequentes são mais complexos e diversificados em suas implementações. A análise é o passo mais custoso quanto aos recursos de processamento. A pontuação comumente é feita de maneira binária: ou um caso de teste passa completamente ou falha completamente (WASIK et al., 2018).

Alguns exemplos de juízes on-line são apresentados da Tabela 2.1.

Nome	URL	Propósito
CodeBench	https://codebench.icomp.ufam.edu.br/	Educacional
Run.codes	https://runcodes.icmc.usp.br/	Educacional
Sphere Online Judge	https://www.spoj.com/	Educacional
LeetCode	https://leetcode.com/	Preparação para entrevistas
beecrowd[2]	https://beecrowd.com/	Competição
Codeforces	https://codeforces.com/	Competição
Online Judge[3]	https://onlinejudge.org/	Competição
HackerRank	https://www.hackerrank.com/	Recrutamento
codejudge	https://www.codejudge.io/	Recrutamento

Tabela 2.1: Alguns exemplos de juízes on-line.

2.3 Detecção de plágio

Um aspecto importante de juízes on-line é a detecção de plágio. Nessas situações, os usuários que cometeram o plágio devem sofrer alguma penalidade.

Um algoritmo projetado para realizar tal detecção é o *winnowing*, que identifica com precisão cópias completas ou parciais dentro de conjuntos de documentos. Isso pode então ser utilizado para investigar se o plágio existe e qual é sua origem. Algumas propriedades que

[1] Isto significa que as condições computacionais de memória, do sistema operacional, de processamento, entre outras devem ser sempre as mesmas. Dessa forma, os recursos computacionais disponíveis serão os mesmos para qualquer caso de teste executado, garantindo assim que não ocorram inconsistências nos seus resultados.

[2] Anteriormente, URI Online Judge.

[3] Anteriormente, UVa Online Judge.

tornam o *winnowing* interessante para sua aplicação em documentos de código fonte são: não é sensível a espaços em branco, possui supressão de ruído, e independe a posição do texto copiado (SCHLEIMER; WILKERSON; AIKEN, 2003).

Um serviço, disponibilizado pela Universidade Stanford, que detecta a similaridade entre códigos é o *Measure Of Software Similarity* (MOSS). Foi criado com o propósito de ajudar na detecção de plágio em turmas de programação. O MOSS utiliza o algoritmo *winnowing* descrito em (SCHLEIMER; WILKERSON; AIKEN, 2003) para gerar um placar de similaridade entre pares de códigos, pares com alto placar de similaridade podem então ser inspecionados por um professor ou monitor da turma para confirmar o plágio (AIKEN, 2010).

2.4 *Learning Analytics*

Recentemente, a pesquisa sobre a análise automatizada de dados educacionais, para fomentar uma melhor experiência de aprendizado, começou a ser chamada de *Learning Analytics* (LA). Apesar de existirem diferentes definições para LA, elas possuem como objetivo em comum a conversão de dados educacionais em ações úteis para fomentar o aprendizado (CHATTI et al., 2012).

As ferramentas utilizadas no EaD, cujo uso cresceu significativamente nos últimos anos, possuem a capacidade de coletar dados de alta qualidade e em grandes quantidades para aplicações em LA. Esses dados são cada vez mais armazenados por entidades educacionais e posteriormente utilizados em análises de LA(BAKER; INVENTADO, 2014).

A granularidade, no contexto da coleta de dados em um sistema de LA, se refere à frequência de coleta dos dados e ao detalhe coletado em cada coleta (JADUD, 2006a). A granularidade dos dados coletados é variável. Em (JADUD, 2006b), por exemplo, o algoritmo apresentado baseia-se no código fonte capturado no momento em que o usuário executa a compilação, equivalente à submissão em um juiz on-line. Em contrapartida, em (PEREIRA et al., 2020), cada pressionamento de tecla e clique de mouse do usuário com o editor é capturada. Espera-se que sistemas com maior granularidade de coleta permitem um *feedback* mais refinado aos estudantes (PEREIRA et al., 2020).

O juiz on-line apresentado neste trabalho tem como um de seus objetivos a coleta de dados de pressionamento de tecla e clique de mouse dos usuários, para uso futuro em pesquisas em LA.

2.5 Considerações sobre a segurança

Compilar e executar qualquer código enviado por um usuário em um servidor pode apresentar grandes riscos de segurança, que devem ser considerados ao se projetar a arquitetura de um juiz on-line de código (WASIK et al., 2018). Portanto, medidas precisam ser tomadas para isolar os processos que executam os códigos enviados por usuários do restante do sistema. Isso garante a estabilidade do servidor, e a segurança e integridade dos seus dados.

Uma técnica muito utilizada para mitigar ataques vindos de entradas não confiáveis é a virtualização (ZHANG, 2022; GOLDBERG, 1973), uma tecnologia importante para a computação em nuvem (OGU et al., 2014) que oferece uma reprodução fiel de um sistema computacional completo onde são executadas aplicações. Vantagens da virtualização incluem o aumento da robustez, da segurança e da privacidade nos sistemas que a utilizam (GOLDBERG, 1973).

Contêineres são uma tecnologia mais recente de virtualização, e virtualizam no nível do sistema operacional, enquanto tecnologias mais tradicionais, baseadas em hipervisores (softwares que criam e executam máquinas virtuais), executam os sistemas virtualizados em um hard-

ware emulado. Tal abordagem abre mão de um nível mais alto de isolamento entre sistemas para ganhar mais desempenho (MERKEL, 2014). Quando comparados com máquinas virtuais tradicionais, executadas em um hipervisor, geralmente apresentam um desempenho superior (MORABITO; KJÄLLMAN; KOMU, 2015).

O Docker é um exemplo de software de conteinerização para a plataforma Linux, utilizando-se de recursos disponibilizados pelo seu *kernel*, muito utilizado devido a seu foco em fornecer desempenho e segurança em diversos ambientes de produção e desenvolvimento (MERKEL, 2014). Outros exemplos incluem o Linux Container (LXC) e o CoreOS rocket (VERMA; PANDEY; GUPTA, 2023).

Capítulo 3

Trabalhos relacionados

Ala-Mutka em (ALA-MUTKA, 2005) apresenta uma pesquisa sobre as abordagens utilizadas por sistemas de correção automática de código. A pesquisa discute o uso de sistemas de correção automática de código no contexto da educação superior, focando na sua viabilidade e as funcionalidades esperadas de tais ferramentas. A autora argumenta que "cursos de programação estão bem posicionados para desenvolver e utilizar ferramentas para a avaliação automatizada de exercícios". Também classifica os sistemas de correção automática de código baseado nos seus métodos de funcionamento, como os de avaliação dinâmica e aferição estática.

Em (CARVALHO BRUNO FREITAS GADELHA, 2016), os autores exploram o uso de juízes on-line no contexto do ensino híbrido de programação na Universidade Federal do Amazonas (UFAM). O ensino híbrido descrito no trabalho consiste na combinação de aulas presenciais com atividades disponibilizadas em um juiz on-line chamado CodeBench, de sua própria autoria. O trabalho comparou turmas onde a metodologia de ensino híbrida foi aplicada com turmas de controle, e concluiu que a metodologia de ensino apresentada aumentou os índices de aprovação das turmas, e motivou o aprendizado e autonomia dos alunos. Entre os projetos pesquisados, o CodeBench é o que mais se assemelha ao sistema apresentado neste trabalho em termos de objetivos relacionados à edição de código on-line e coleta de dados de interação dos alunos.

A gamificação é definida como a aplicação de componentes de jogos em contextos que não envolvem jogos (DETERDING et al., 2011). O uso de gamificação em exercícios de programação disponibilizados em um juiz on-line é explorado em (RODRIGUES et al., 2022). No estudo, foram utilizadas duas versões do juiz on-line CodeBench: uma versão modificada para incluir elementos de gamificação e outra, sem modificações, usada pelo grupo de controle. Os resultados indicaram que o efeito positivo da gamificação diminui inicialmente, mas aumenta novamente de forma natural após um certo intervalo de tempo.

Por se tratar de um juiz on-line, os requisitos levantados no projeto desenvolvido neste Trabalho de Conclusão de Curso seguem parcialmente os descritos por (WATANOBE et al., 2022). Nesse estudo, os autores identificam requisitos funcionais e não funcionais para o desenvolvimento de um juiz on-line genérico e apresentam a arquitetura de um sistema implementado por eles, ilustrada na Figura 3.1. Além disso, relata as experiências e desafios encontrados durante o seu desenvolvimento.

Uma arquitetura alternativa, distribuída, é apresentada em (WANG; HAN; CHEN, 2021). Os autores implementaram um juiz on-line o qual chamaram de MetaOJ, através da modificação de um sistema de arquitetura não distribuída existente chamado TUOJ. Experimentos foram realizados para comparar o desempenho entre o MetaOJ e o TUOJ, mas os autores não observaram mudanças significativas no desempenho. Segundo os autores, as vantagens da

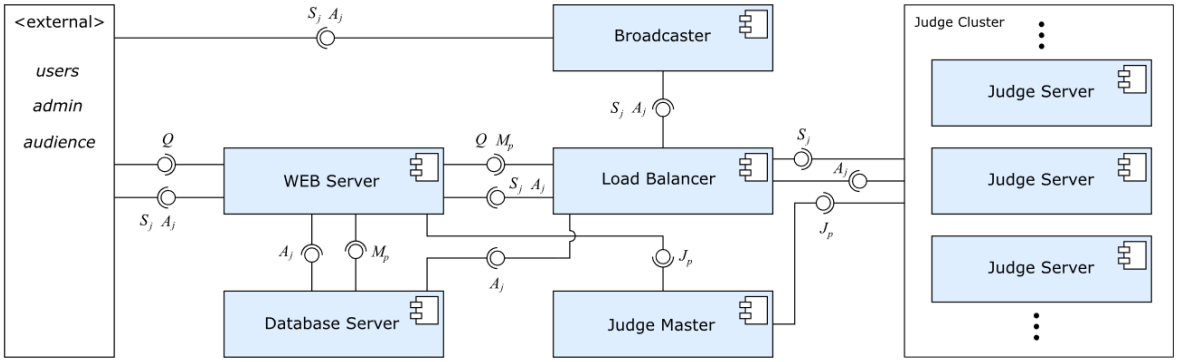

Figura 3.1: Arquitetura de um juiz on-line. Fonte: (WATANOBE et al., 2022)

arquitetura são a tolerância a falhas e facilidade de escalonamento.

Em (MÉSZÁROS; CSERÉP; FEKETE, 2019), os autores propõem o uso do Language Server Protocol (LSP) como uma abordagem para incorporar funcionalidades típicas de IDEs em diferentes tipos de software. Nesse trabalho, assim como neste TCC, o LSP foi utilizado para adicionar funcionalidades a um sistema web. No entanto, enquanto o trabalho apresentado foca em uma ferramenta para navegação em código-fonte em grandes bases de código, neste TCC o LSP foi empregado na implementação de um editor de código voltado para a web.

A ideia de coletar dados sobre a codificação de alunos em cursos introdutórios de programação existe há decadas, e foi demonstrada em (SPOHRER; POPE, 1985). Nesse trabalho, os erros cometidos pelos estudantes foram analisados e classificados pelos seus tipos. Em suas conclusões, os autores sugeriram a automatização do processo de análise, proposta que está alinhada com o funcionamento de juízes on-line, incluindo o sistema desenvolvido neste trabalho.

Referente à coleta de grande quantidade de dados de usuários de uma IDE, em (BROWN et al., 2014), o autor apresenta um projeto educational de coleta de dados de programação chamado Blackbox. Os dados são coletados de usuários do BlueJ IDE, um ambiente de programação Java (KöLLING; ROSENBERG, 1999). Esses dados contém o comportamento de edição do código fonte dos usuários, coletados a cada linha de código editada. No artigo, são apresentados os desafios enfrentados na coleta de dados, são apresentados os dados coletados pelo Blackbox, e são feitas análises de uma amostra dos dados, como exemplos de como os dados podem ser utilizados para pesquisa. Para que os dados sejam enviados para o projeto, os usuários devem autorizar a coleta. De acordo com o autor 42% dos usuários autorizaram a coleta. Nos 6 primeiros meses o projeto contava com mais de 150.000 participantes e foram coletados dados de mais de 10.000.000 compilações.

(PEREIRA et al., 2020) apresenta a aplicação de *Learning Analytics* em estudantes de cursos introdutórios de programação. No trabalho, estudantes da Universidade Federal do Amazonas (UFAM) utilizaram a ferramenta juiz on-line CodeBench em suas atividades. O CodeBench coletou dados de interação de 2058 estudantes com o sistema, abrangendo 2.058 alunos de cursos não relacionados à Ciência da Computação, provenientes de turmas desde 2016. Entre os dados coletados estão os pressionamentos do teclado e do mouse. Na Figura 3.2, é apresentado um exemplo dos dados coletados em formato de *logs*, retirado do *dataset*[1]. O trabalho visou responder duas questões de pesquisa:

(a) Como comportamentos eficazes ou ineficazes podem ser identificados rapidamente?

(b) Quais comportamentos eficazes podem ser utilizados para guiar estudantes com compor-

[1]https://codebench.icomp.ufam.edu.br/dataset/

tamentos ineficazes?

Os autores identificaram 3 tipos diferentes de comportamento entre os estudantes, com diferentes níveis de eficácia. Os autores sugeriram que essa identificação pode ajudar nas decisões pedagógicas dos professores.

Figura 3.2: Exemplo de dados coletados pelo CodeBench, onde o usuário navega e digita `print`.

```
1 2016-6-2 19:48:56.409#keyHandled#"Down"
2 2016-6-2 19:48:56.958#keyHandled#"Left"
3 2016-6-2 19:48:57.954#change#{"from":{"line":1,"ch":0},"to":{"line":1,"ch":0},
    "text":["p"],"removed":[""],"origin":"+input"}
4 2016-6-2 19:48:58.220#change#{"from":{"line":1,"ch":1},"to":{"line":1,"ch":1},
    "text":["r"],"removed":[""],"origin":"+input"}
5 2016-6-2 19:48:58.432#change#{"from":{"line":1,"ch":2},"to":{"line":1,"ch":2},
    "text":["i"],"removed":[""],"origin":"+input"}
6 2016-6-2 19:48:58.658#change#{"from":{"line":1,"ch":3},"to":{"line":1,"ch":3},
    "text":["n"],"removed":[""],"origin":"+input"}
7 2016-6-2 19:48:58.852#change#{"from":{"line":1,"ch":4},"to":{"line":1,"ch":4},
    "text":["t"],"removed":[""],"origin":"+input"}
```

No sistema apresentado neste trabalho, a coleta de dados dos alunos ocorre tanto no momento de submissão — equivalente à coleta durante a compilação, conforme descrito em (JADUD, 2006b) — quanto no editor de código web, durante interações dos alunos, seguindo uma abordagem similar à utilizada no juiz online CodeBench, descrita em (PEREIRA et al., 2020).

Capítulo 4

Resultados

Este capítulo apresenta os artefatos construídos nas etapas do processo de desenvolvimento de software ICONIX. Neste trabalho, a modelagem de apenas um subconjunto de funcionalidades mais importantes será detalhada.

4.1 Análise de requisitos

Os artefatos resultantes da etapa de análise de requisitos incluem o diagrama de classes de alto nível e o diagrama de casos de uso. O diagrama de classes de alto nível é ilustrado na Figura 4.1. O diagrama de casos de uso é apresentado parcialmente nesta seção, na Figura 4.2, e de forma completa no Apêndice A.

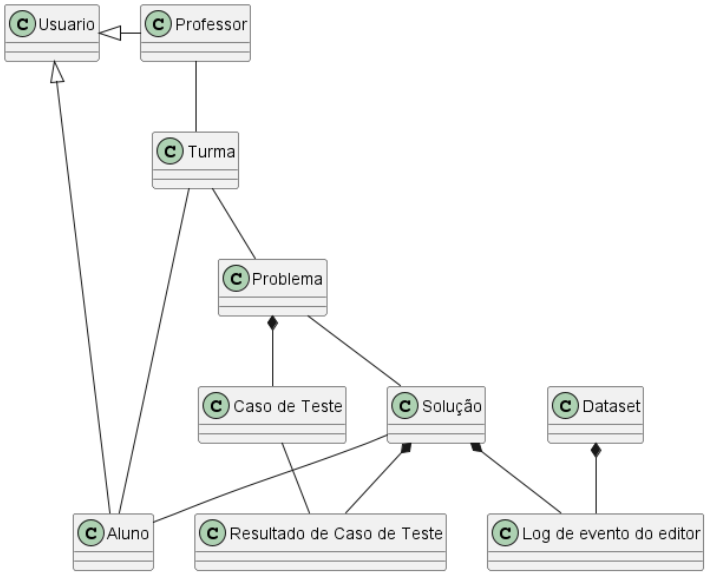

Figura 4.1: Diagrama de domínio.

4.1.1 Requisitos funcionais

1. **RF-001** Usuários devem ser capazes de fazer login no sistema.
2. **RF-002** Usuários devem ser capazes de se cadastrar através do e-mail institucional da UEA.
3. **RF-003** Professores devem ser capazes de criar turmas para suas disciplinas.

4. **RF-004** O sistema deve atribuir um código aleatório único para turmas novas.
5. **RF-005** Professores devem ser capazes de solicitar um novo código aleatório para uma turma.
6. **RF-006** Professores devem ser capazes de criar problemas, associados a uma turma.
7. **RF-007** O sistema deve permitir que alunos ingressem em uma turma utilizando o código da turma.
8. **RF-008** O sistema deve permitir que o aluno visualize as turmas onde está matriculado
9. **RF-009** O sistema deve permitir que usuários usem o editor para escrever e enviar para o *backend* uma solução para um problema, que será avaliada.
10. **RF-010** O sistema deve testar as soluções enviadas e mostrar ao aluno a sua pontuação no problema.
11. **RF-011** O sistema deve listar as pontuações de todos alunos em todos os problemas de uma turma para o professor da turma.
12. **RF-012** O sistema deve gerar um arquivo de *log* com os eventos de interação do aluno com o editor de texto e com a IU do sistema.

4.1.2 Requisitos não funcionais

1. **RNF-001**. O sistema deve compilar e executar os casos de teste em contêineres Docker para oferecer segurança.
2. **RNF-002** A interface *web* do sistema deve ser acessível através da internet.
3. **RNF-003** A interface *web* do sistema deve ser utilizável em computadores pessoais.
4. **RNF-004** A interface *web* do sistema deve ser utilizável nos navegadores Google Chrome a partir da versão 115 (Google, 2008) e Mozilla Firefox a partir da versão 115.0 (Mozilla Foundation, 2004).

4.2 Análise e projeto preliminar

Esta seção apresenta os artefatos construídos na etapa de análise e projeto preliminar do ICONIX. A partir dos casos de uso identificados anteriormente, os caminhos base e alternativos foram identificados para cada um deles, são enumerados a seguir os caminhos referentes ao subconjunto de casos de uso apresentados na Figura 4.2.

Os requisitos funcionais levantados estão alocados nos casos de uso a seguir.

4.2.1 Caso de uso CDU-1: Aluno ingressa em uma turma utilizando o código da turma

Este caso de uso se refere ao requisito funcional RF-007: o sistema deve permitir que alunos ingressem em uma turma utilizando o código da turma. Esse código é gerado previamente pelo professor e enviado aos alunos. O diagrama de robustez está apresentado na Figura 4.3.

> **Caminho base**: O sistema apresenta a página de turmas ao aluno. O aluno insere o código da turma no campo de ingresso em turma e aperta o botão para ingressar. O sistema valida o código e, caso seja válido, insere o aluno na turma correspondente. Em seguida, o sistema exibe na página a turma na qual o aluno ingressou.

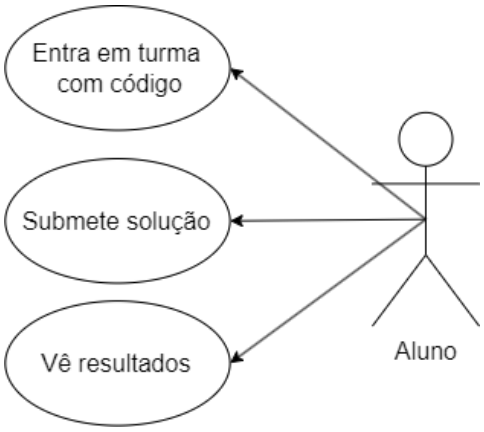

Figura 4.2: Diagrama de caso de uso, apresentando um subconjunto de funcionalidades.

Caminho alternativo: Se o código for inválido, uma mensagem de erro é exibida ao aluno informando que o código inserido é inválido.

4.2.2 Caso de uso CDU-2: Aluno envia a solução de um problema

Este caso de uso se refere ao requisito funcional RF-009: o sistema deve permitir que alunos usem o editor web para escrever e enviar uma solução ao sistema de avaliação do juiz online. O sistema, então, testa a solução enviada. O diagrama de robustez está apresentado na Figura 4.4.

Caminho base: O sistema apresenta a página que contém o editor de código. O aluno implementa a solução do problema e aperta o botão de enviar solução. O sistema recebe a solução e a armazena no banco de dados, inserindo-a ao final da fila de execução de testes. O aluno é redirecionado para a página do problema, onde o sistema mostra uma mensagem indicando que sua solução está esperando execução.

Caminho alternativo: Se o código fonte enviado causar um erro de compilação,a solução não é testada e uma mensagem erro é exibida informando ao aluno o problema ocorrido durante a compilação.

4.2.3 Caso de uso CDU-3: Aluno vê os resultados das suas soluções

Este caso de uso se refere ao requisito funcional RF-010: o sistema deve fornecer os resultados das soluções enviadas pelo aluno para um problema, incluindo a pontuação obtida. O diagrama de robustez está apresentado na Figura 4.5.

Caminho base: O sistema apresenta a página do problema para o aluno. O sistema acessa o banco de dados e busca as soluções enviadas pelo aluno. Cada envio de solução encontrado é listado para o aluno, com dados do envio: data de envio, estado atual de execução (em espera, executando, avaliado, erro), e caso a execução esteja finalizada, a pontuação obtida e tempo de execução do programa.

4.2.4 Análise de robustez

Com base nos casos de uso descritos, foi realizada a análise de robustez. Para cada caso de uso, criou-se o seu respectivo diagrama de robustez. Os diagramas referentes aos RF-007, RF-009 e RF-010 são apresentados nas figuras desta seção[1]. Os diagramas de robustez restantes são

[1]No ICONIX, todos os diagramas devem ter em sua proximidade os textos dos caminhos identificados nos casos de uso. Os casos de uso deste trabalho foram descritos na seção anterior (4.2). Portanto, os textos dos

apresentados no apêndice B.

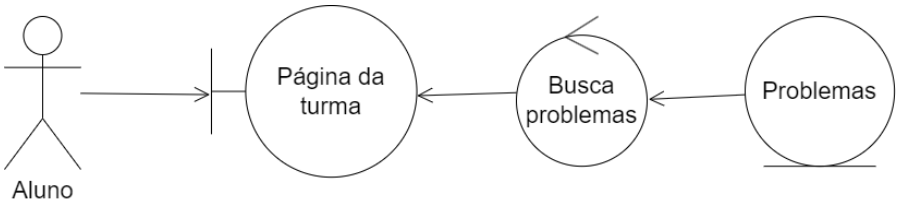

Figura 4.3: Diagrama de robustez para RF-007

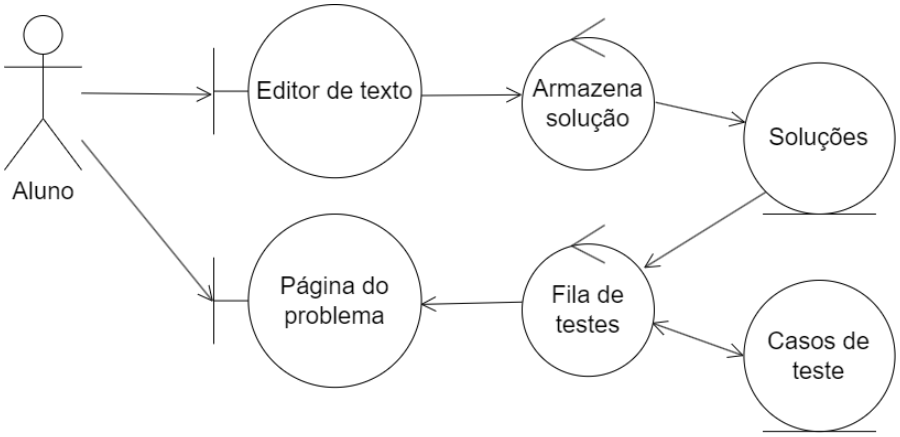

Figura 4.4: Diagrama de robustez para RF-009

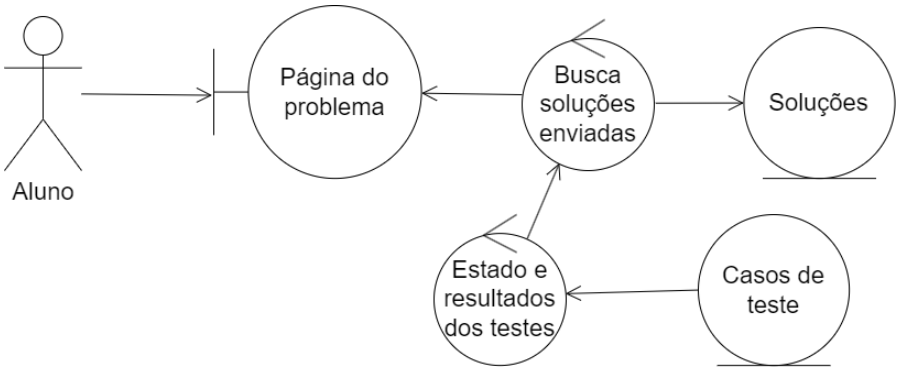

Figura 4.5: Diagrama de robustez para RF-010

4.3 Projeto detalhado

Nesta etapa do processo ICONIX, os diagramas de sequência foram feitos, baseando-se na análise robusta realizada na etapa anterior.

O diagrama de sequência na Figura 4.6, referente ao requisito funcional RF-007, ilustra a interação entre:

(a) O ator “Aluno”;

(b) A interface “View de turmas”;

(c) O modelo de dado “Model de turmas”.

caminhos foram omitidos dessa seção para evitar repetição de texto e para explicitar a sequência das etapas do processo.

Nesta interação, o aluno utiliza a página web para se matricular por código em uma das turmas, fazendo uma requisição à "View de turmas", que consulta o "Model de turmas" para verificar se o código corresponde a uma turma que existe no bando de dados. Em caso afirmativo, o aluno é matriculado na turma correspondente ao código e a "View de turmas" redireciona o usuário para a página da turma recém matriculada. Já no caso em que o código não corresponde a uma turma que existe, uma mensagem de erro é retornada ao aluno e exibida na página web ao qual ele é redirecionado.

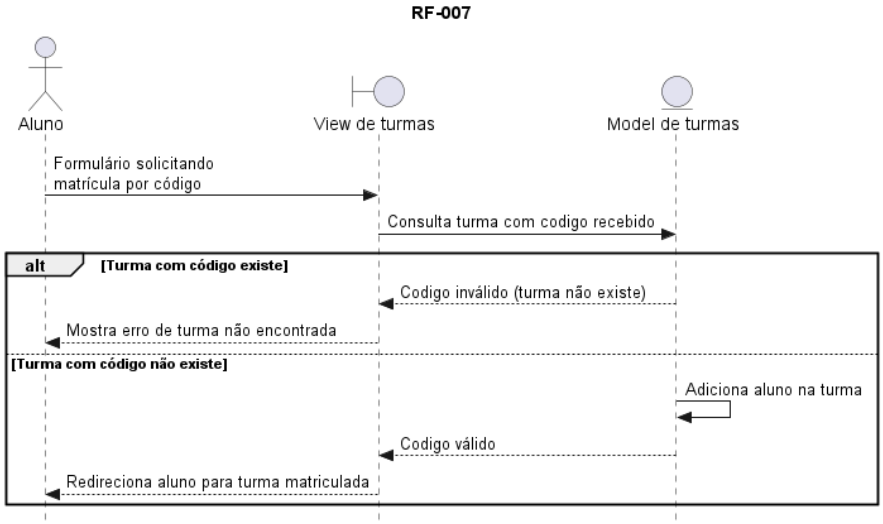

Figura 4.6: Diagrama de sequência RF-007

O diagrama de sequência na Figura 4.7, referente ao requisito funcional RF-009, ilustra a interação entre:

(a) O ator "Aluno";

(b) As interfaces "Editor de texto" e "View de submissão";

(c) O modelo de dado "Model de submissão".

Nesta interação, o aluno utiliza o editor web para codificar uma solução para um problema. O aluno então submete a sua solução, apertando uma botão presente na tela do editor. O editor se comunica com a "View de submissão" através de uma requisição HTTP, enviando os dados da submissão. Os dados são salvos no bando de dados através do "Model de submissão" e então a "View de submissão" redireciona o aluno para a página de resultados do problema.

O diagrama de sequência na Figura 4.8, referente ao requisito funcional RF-010, ilustra a interação entre:

(a) O ator "Aluno";

(b) A interface"View de problemas";

(c) O objeto de controle "Testador de submissões";

(d) Os modelos de dados "Model de problemas" e "Model de submissões".

Nesta interação, um loop executa constantemente em um fio de execução separado dos outros do sistema, nele o objeto "Testador de submissões" verifica constantemente se existem objetos de submissão não testados. Ao encontrar sobmissões não testadas, o "Testador de submissões" as testa com os casos de teste registrados no objeto de problema ligado à submissão, registrando nas submissões os resultados dos casos de teste.

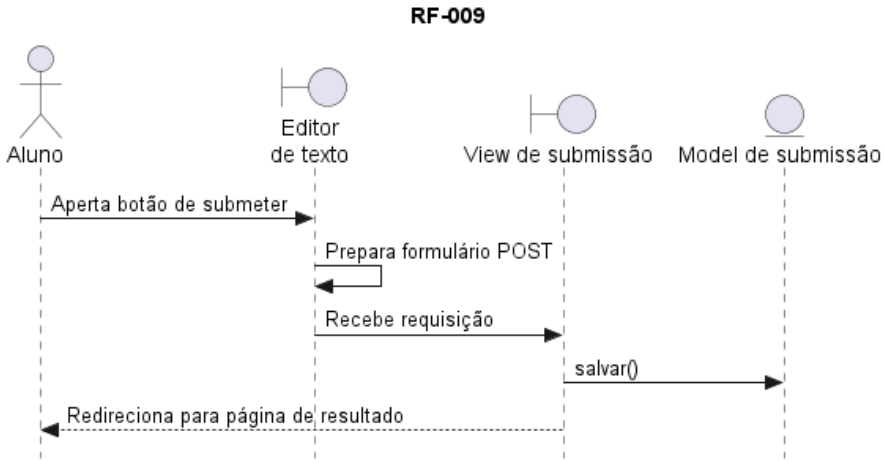

Figura 4.7: Diagrama de sequência RF-009

O usuário utiliza a página web para acessar os resultados, fazendo uma requisição à "View de problemas", que consulta o "Model de problemas" e "Model de submissões" para retornar à página os resultados do aluno de suas submissões.

Atualizando o diagrama de domínio com atributos e métodos, obtemos o diagrama de classe completo, apresentado na Figura 4.9.

4.4 Implementação

Seguindo o que foi modelado no processo ICONIX, um protótipo do sistema proposto foi desenvolvido, implementando os requisitos funcionais levantados.

4.4.1 Arquitetura do sistema

A arquitetura do protótipo é apresentada no diagrama da Figura 4.10. Nele, estão separados os componentes do *backend* e *frontend*.

O administrador interage com a área de administração do Django, enquanto os usuários apenas interagem com os outros componentes do *frontend*. Entre o editor no *frontend* e o servidor LSP no *backend*, existe uma conexão *WebSocket*. Todos os outros componentes do *frontend* se comunicam com o *backend* através de requisições HTTP. Os eventos de interação dos usuários com o editor são coletados por um módulo JavaScript, e enviados para o Django.

Para garantir uma alocação de recursos computacionais consistente, os programas enviados pelos usuários são inseridos em uma fila de execução no banco de dados. Uma *thread* separada consome os elementos dessa fila, de forma que apenas um programa esteja em execução a cada momento, evitando a concorrência de recursos computacionais.

4.4.2 Amostra da interface do sistema

Ao acessar um dos problemas, o usuário é levado para uma página contendo o enunciado do problema, datas relevantes do problema e arquivos de suporte anexados. A página é mostrada nas Figura 4.11 e Figura 4.12.

Através do link de abrir editor *web*, mostrado na Figura 4.12, o usuário é redirecionado para o editor de código online, mostrado na Figura 4.13, onde pode codificar e enviar um programa que resolva o problema proposto.

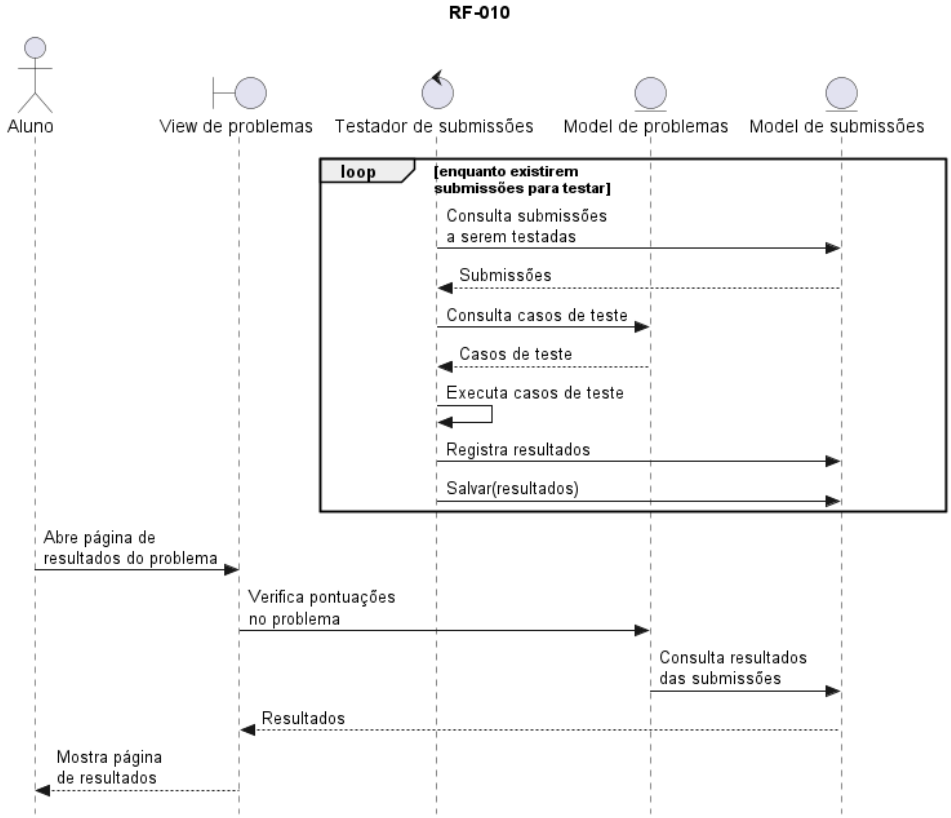

Figura 4.8: Diagrama de sequência RF-010

Após realizar a submissão do programa criado, ele é recebido pelo servidor, compilado e executado com as entradas dos casos de teste do problema. Os resultado dos casos de teste para cada envio são então exibidos na página do problema, na área de submissões mostrada na Figura 4.14.

Na Figura 4.15 e na Figura 4.16 são apresentados a lista de eventos de editor capturados, e os detalhes de um evento, respectivamente. Na Figura 4.15 é possível ver os diferentes tipos de evento que são capturados. Na Figura 4.16 é possível visualizar os detalhes de um evento de tipo mudança de conteúdo de texto no editor, nos seus detalhes temos o conteúdo inserido, a data e hora do evento para o cliente, a posição onde foi inserido, o usuário que fez a edição, e o problema associado ao evento.

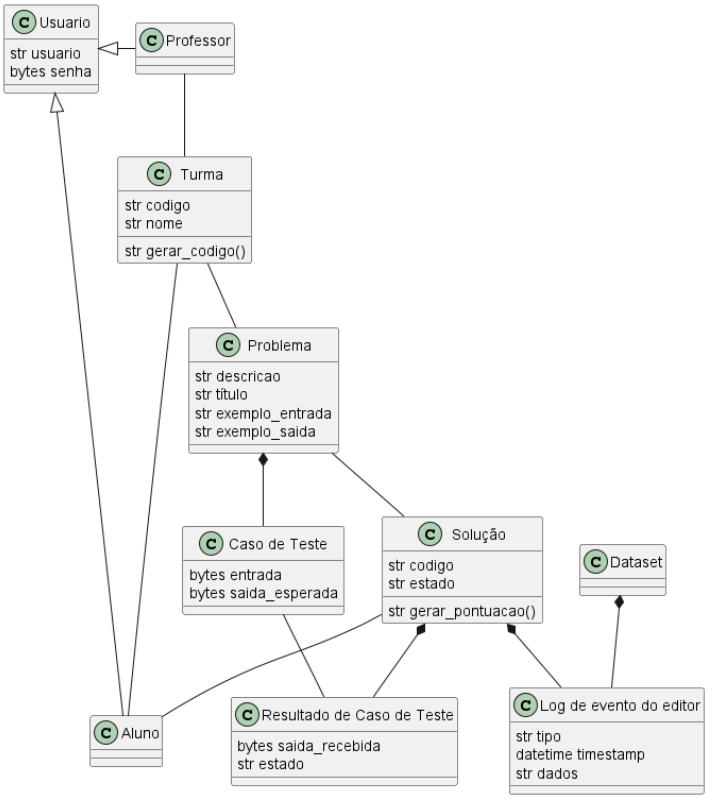

Figura 4.9: Diagrama de classe.

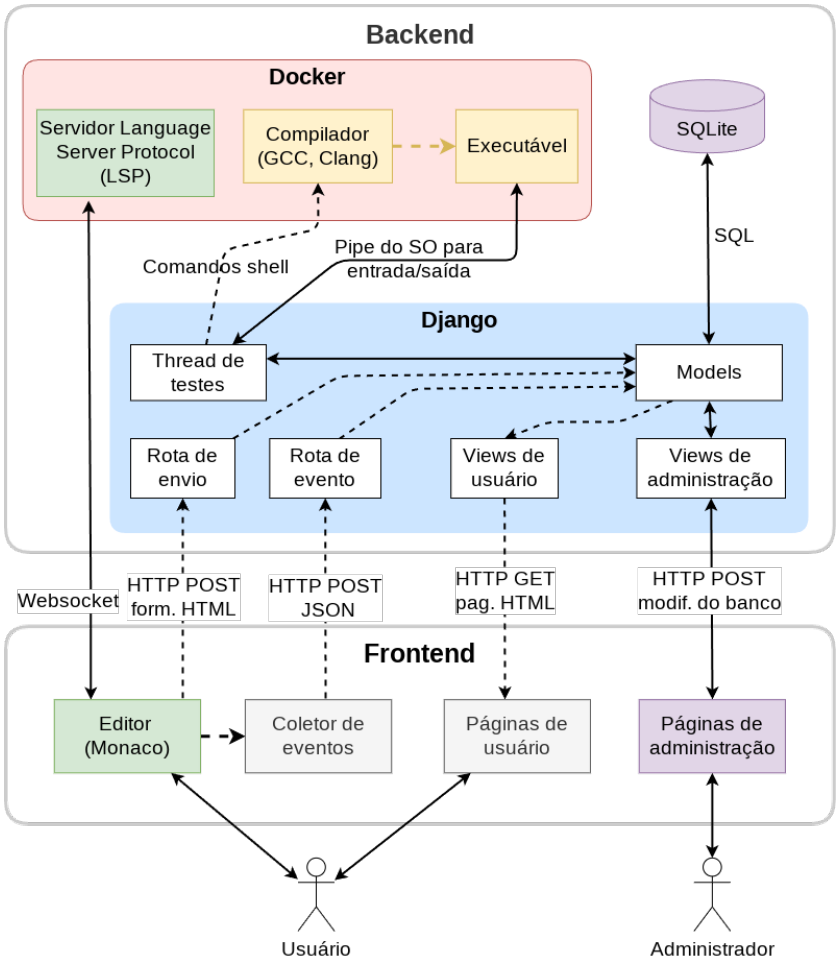

Figura 4.10: Diagrama de arquitetura

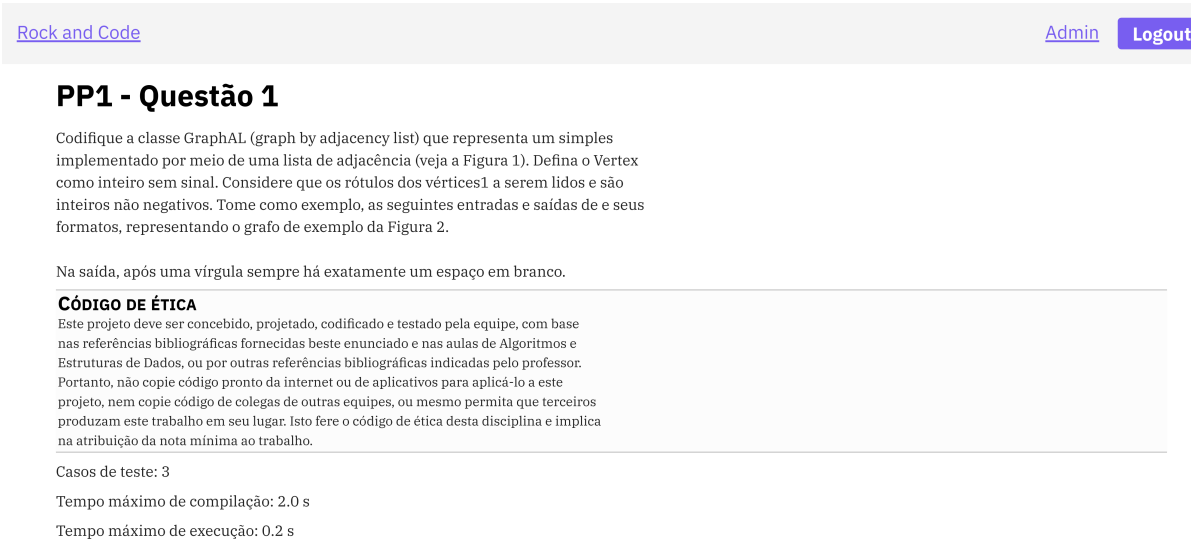

Figura 4.11: Página de um problema, mostrando seu enunciado.

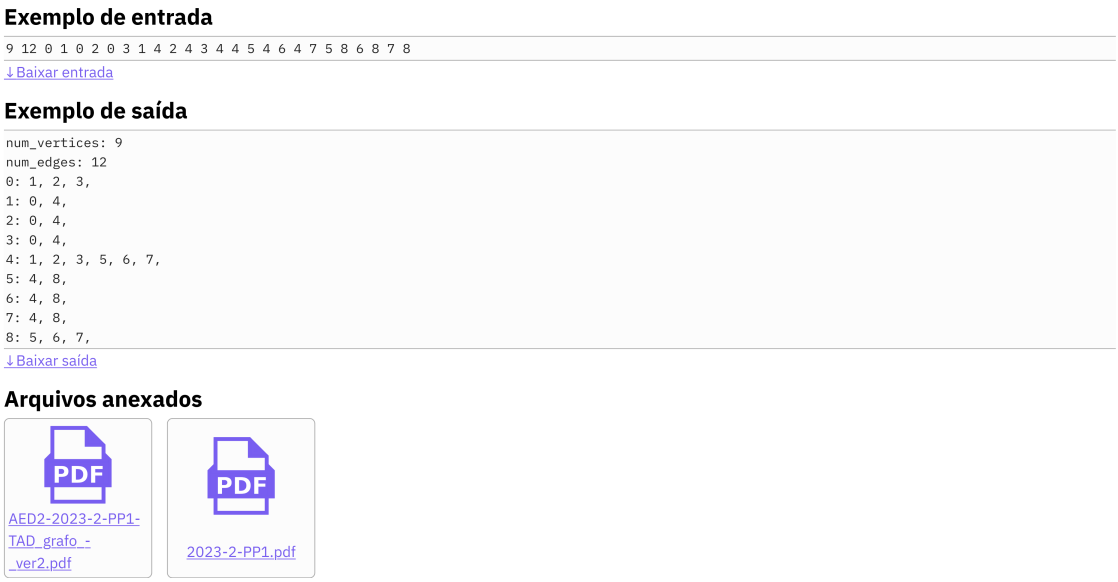

Figura 4.12: Página de um problema, mostrando entrada, saída, arquivos anexados e datas.

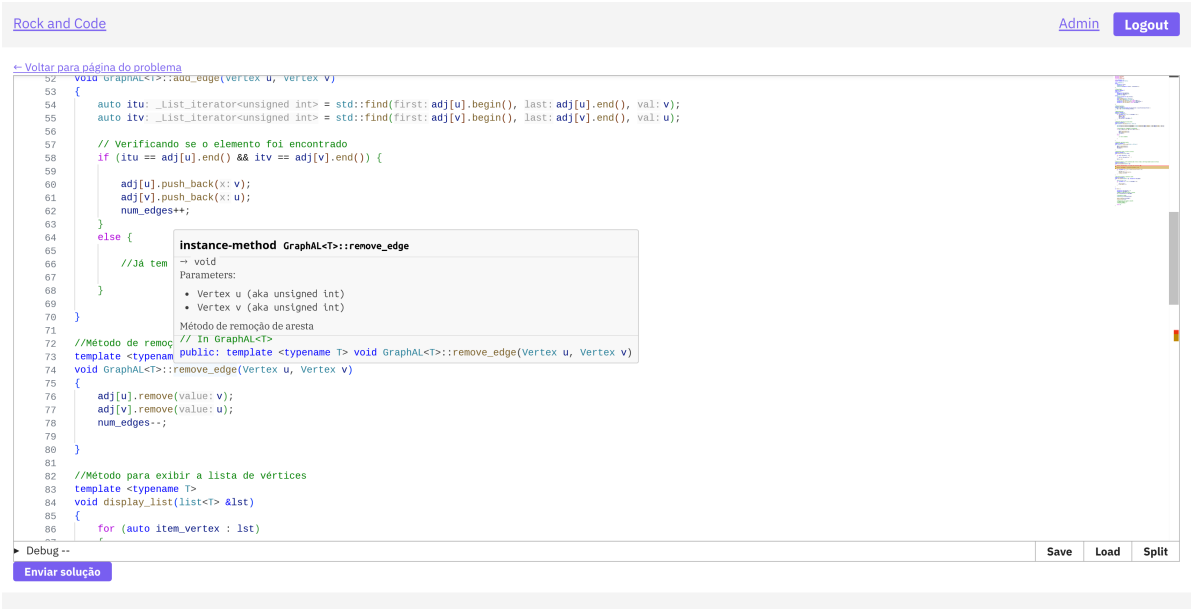

Figura 4.13: Página do editor, mostrando dentro dele o código e funcionalidades de coloração de sintaxe, documentação e análise de símbolos.

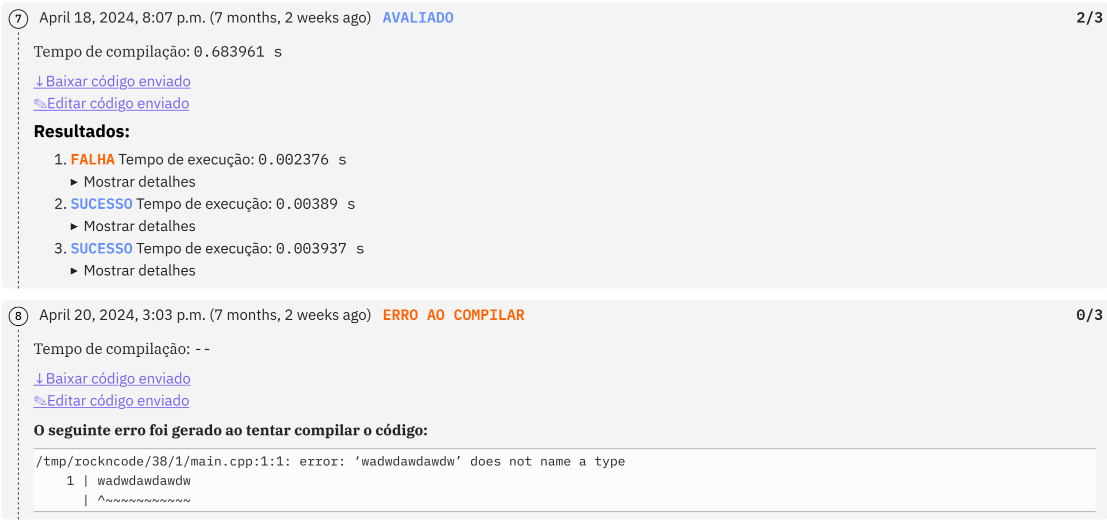

Figura 4.14: Área de submissões da página de um problema.

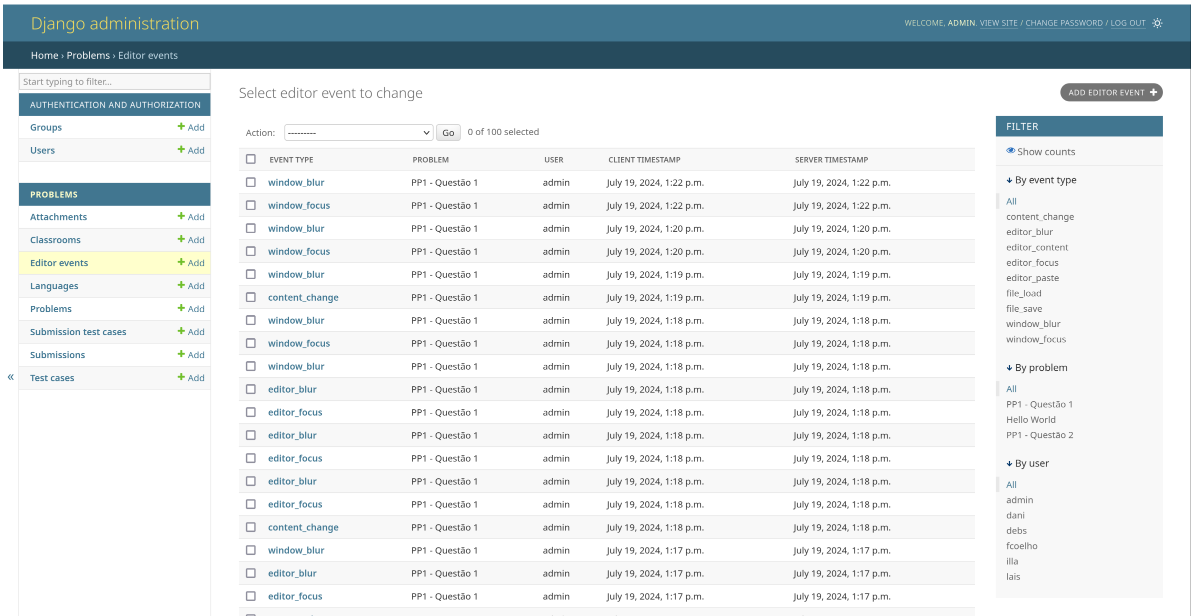

Figura 4.15: Página de administração mostrando lista de eventos de edição armazenados no banco de dados.

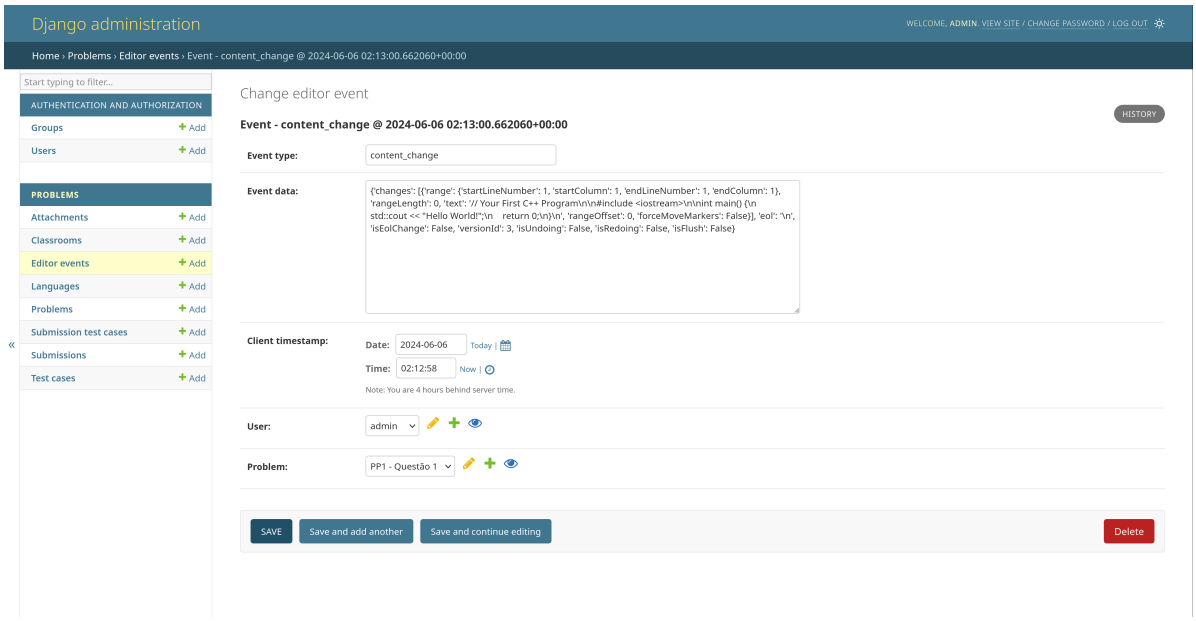

Figura 4.16: Página mostrando detalhes de um único evento de edição.

Capítulo 5

Considerações finais

O presente Trabalho de Conclusão de Curso teve como objetivo desenvolver um juiz on-line educacional para auxiliar professores e alunos em disciplinas introdutórias de programação da Escola Superior de Tecnologia da Universidade do Estado do Amazonas (UEA).

A metodologia de desenvolvimento utilizada foi o processo ICONIX, que compreende as etapas de análise de requisitos, análise e projeto preliminar, projeto detalhado e implementação. Essa abordagem permitiu estruturar de forma sistemática o desenvolvimento do sistema proposto.

No decorrer do processo de desenvolvimento, foram definidos os casos de uso, levantados os requisitos, realizadas a análise de robustez e o projeto preliminar. Com base nos artefatos gerados, o sistema foi implementado, priorizando as funcionalidades consideradas mais relevantes. No entanto, devido a limitações de tempo e infraestrutura, algumas funcionalidades, como o uso completo de contêineres Docker, a detecção de plágio e a criação de um banco de problemas ficarão para trabalhos futuros.

5.1 Trabalhos futuros

Como continuidade deste trabalho, estão planejadas as seguintes ações e pesquisas futuras:

- Hospedar a ferramenta em um servidor da UEA e disponibilizá-la para uso pelos professores.
- Realizar uma análise de segurança do sistema para garantir sua confiabilidade e proteção contra possíveis vulnerabilidades.
- Criar um banco de questões que possa ser compartilhado entre diferentes turmas e professores, facilitando a reutilização de exercícios
- Analisar os efeitos do uso do juiz on-line no processo de aprendizado dos alunos da UEA.
- Utilizar o *dataset* coletado das interações dos alunos em pesquisas de *Learning Analytics*, contribuindo para a melhoria do ensino de programação.

Bibliografia

AIKEN, A. *MOSS*. 2010. <https://theory.stanford.edu/~aiken/moss/>. Acesso em: 2024-07-20.

ALA-MUTKA, K. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, v. 15, p. 83–102, 06 2005.

BAKER, R. S.; INVENTADO, P. S. Educational data mining and learning analytics. In: _____. *Learning Analytics: From Research to Practice*. New York, NY: Springer New York, 2014. p. 61–75. ISBN 978-1-4614-3305-7. Disponível em: <https://doi.org/10.1007/978-1-4614-3305-7_4>.

BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Change (2nd Edition)*. [S.l.]: Addison-Wesley Professional, 2004. ISBN 0321278658.

BLINOWSKI, G.; OJDOWSKA, A.; PRZYBYŁEK, A. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, v. 10, p. 20357–20374, 2022.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. Unified modeling language user guide, the (2nd edition) (addison-wesley object technology series). *J. Database Manag.*, v. 10, 01 1999.

BROWN, N. C. C. et al. Blackbox: a large scale repository of novice programmers' activity. In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2014. (SIGCSE '14), p. 223–228. ISBN 9781450326056. Disponível em: <https://doi.org/10.1145/2538862.2538924>.

CARVALHO BRUNO FREITAS GADELHA, D. F. L. G. Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In: *XXVII Simpósio Brasileiro de Informática na Educação*. [S.l.: s.n.], 2016. p. 140.

CARVALHO, L. S. G.; OLIVEIRA, D. F.; GADELHA, B. Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In: *Proceedings of the XXVII Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação – SBIE 2016)*. [S.l.]: SBC, 2016. p. 140–149. ISSN 2316–6533.

CHATTI, M. A. et al. A reference model for learning analytics. *International Journal of Technology Enhanced Learning*, v. 4, p. 318–331, 2012. Disponível em: <https://api.semanticscholar.org/CorpusID:42326828>.

COMBÉFIS, S. Automated code assessment for education: Review, classification and perspectives on techniques and tools. *Software*, v. 1, n. 1, p. 3–30, 2022. ISSN 2674-113X. Disponível em: <https://www.mdpi.com/2674-113X/1/1/2>.

DETERDING, S. et al. Gamification: Using game design elements in non-gaming contexts. In: . [S.l.: s.n.], 2011. v. 66, p. 2425–2428.

Django Software Foundation. *Django*. 2005. <https://www.djangoproject.com/>. Acesso em: 2024-03-29. Disponível em: <https://www.djangoproject.com/>.

Django Software Foundation. *FAQ: General | Django documentation | Django*. 2005. <https://docs.djangoproject.com/en/5.0/faq/general/>. Accessed: 2024-04-01.

FORSYTHE, G. E.; WIRTH, N. Automatic grading programs. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 8, n. 5, p. 275–278, maio 1965. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/364914.364937>.

Free Software Foundation. *GCC (GNU Compiler Collection)*. 1987. <https://gcc.gnu.org/>. Acesso em: 2024-03-29. Disponível em: <https://gcc.gnu.org/>.

GOLDBERG, R. P. Architecture of virtual machines. In: *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*. New York, NY, USA: Association for Computing Machinery, 1973. (AFIPS '73), p. 309–318. ISBN 9781450379168. Disponível em: <https://doi.org/10.1145/1499586.1499669>.

Google. *Google Chrome*. 2008. <https://www.google.com/chrome/>. Acesso em: 2024-04-11.

HAVERBEKE, M. *CodeMirror*. 2007. <http://codemirror.net/>. Accessed: 2024-04-01. Disponível em: <http://codemirror.net/>.

JADUD, M. C. *An exploration of novice compilation behaviour in BlueJ*. Tese (Doutorado) — University of Kent at Canterbury, 2006. Disponível em: <https://api.semanticscholar.org/CorpusID:59748115>.

JADUD, M. C. Methods and tools for exploring novice compilation behaviour. In: *Proceedings of the Second International Workshop on Computing Education Research*. New York, NY, USA: Association for Computing Machinery, 2006. (ICER '06), p. 73–84. ISBN 1595934944. Disponível em: <https://doi.org/10.1145/1151588.1151600>.

JSON-RPC Working Group. *JSON-RPC 2.0 Specification*. 2013. <https://www.jsonrpc.org/specification>. Acesso em: 2024-07-16.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2004. (Addison-Wesley object technology series). ISBN 9780321197702. Disponível em: <https://books.google.com.br/books?id=RYCMx6o47pMC>.

KöLLING, M.; ROSENBERG, J. *BlueJ*. 1999. <https://bluej.org/>. Acesso em: 2024-10-09. Disponível em: <https://bluej.org/>.

MCCABE, T. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2, n. 4, p. 308–320, 1976.

MELNIKOV, A.; FETTE, I. *The WebSocket Protocol*. RFC Editor, 2011. RFC 6455. (Request for Comments, 6455). Disponível em: <https://www.rfc-editor.org/info/rfc6455>.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, v. 2014, n. 239, p. 2, 2014.

MÉSZÁROS, M.; CSERÉP, M.; FEKETE, A. Delivering comprehension features into source code editors through lsp. In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [S.l.: s.n.], 2019. p. 1581–1586.

Microsoft Corporation. *Monaco Editor*. 2013. <https://microsoft.github.io/monaco-editor/>. Acesso em: 2024-03-29. Disponível em: <https://microsoft.github.io/monaco-editor/>.

Microsoft Corporation. *Language Server Protocol*. 2016. <https://microsoft.github.io/

language-server-protocol/>. Acesso em: 2024-03-29. Disponível em: <https://microsoft.github.io/language-server-protocol/>.

Microsoft Corporation. *Language Servers*. 2017. <https://microsoft.github.io/language-server-protocol/implementors/servers/>. Acesso em: 2024-07-17.

MORABITO, R.; KJÄLLMAN, J.; KOMU, M. Hypervisors vs. lightweight virtualization: A performance comparison. In: *2015 IEEE International Conference on Cloud Engineering*. [S.l.: s.n.], 2015. p. 386–393.

Mozilla Foundation, M. *Mozilla Firefox*. 2004. <https://www.mozilla.org/en-US/firefox/new/>. Acesso em: 2024-04-11.

OGU, E. et al. Virtualization and cloud computing: The pathway to business performance enhancement, sustainability and productivity. *International Journal of Business and Economics Research*, v. 3, p. 170–177, 10 2014.

PEREIRA, F. D. et al. Using learning analytics in the amazonas: understanding students' behaviour in introductory programming. *British Journal of Educational Technology*, v. 51, n. 4, p. 955–972, 2020. Disponível em: <https://bera-journals.onlinelibrary.wiley.com/doi/abs/10.1111/bjet.12953>.

RIBEIRO, R. B. S. et al. Gamificação de um sistema de juiz online para motivar alunos em disciplina de programação introdutória. In: *XXIX Simpósio Brasileiro de Informática na Educação (Brazilian Symposium on Computers in Education)*. [S.l.: s.n.], 2018. p. 805.

RODRIGUES, L. et al. Gamification suffers from the novelty effect but benefits from the familiarization effect: Findings from a longitudinal study. *International Journal of Educational Technology in Higher Education*, v. 19, n. 1, p. 13, fev. 2022. ISSN 2365-9440. Disponível em: <https://doi.org/10.1186/s41239-021-00314-6>.

ROSENBERG, D. Introduction to iconix process. In: _____. *Use Case Driven Object Modeling with UML: Theory and Practice*. Berkeley, CA: Apress, 2007. p. 1–20. ISBN 978-1-4302-0369-8. Disponível em: <https://doi.org/10.1007/978-1-4302-0369-8_1>.

ROSENBERG, K. S. D. *Applying use case driven object modeling with UML: an annotated e-commerce example*. USA: Addison Wesley Longman Publishing Co., Inc., 2001. ISBN 0201730391.

SCHLEIMER, S.; WILKERSON, D. S.; AIKEN, A. Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2003. (SIGMOD '03), p. 76–85. ISBN 158113634X. Disponível em: <https://doi.org/10.1145/872757.872770>.

Sourcegraph Inc. *JSON-RPC 2.0 Specification*. 2016. <https://langserver.org/>. Acesso em: 2024-07-17.

SPOHRER, E. S. J. C.; POPE, E. A goal/plan analysis of buggy pascal programs. *Human–Computer Interaction*, Taylor & Francis, v. 1, n. 2, p. 163–207, 1985.

VERMA, S.; PANDEY, B.; GUPTA, B. Containerization and its architectures: A study. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, v. 11, p. 395–409, 06 2023.

WANG, M.; HAN, W.; CHEN, W. Metaoj: A massive distributed online judge system. *Tsinghua Science and Technology*, v. 26, n. 4, p. 548–557, 2021.

WASIK, S. et al. A survey on online judge systems and their applications. *ACM Comput.*

Surv., Association for Computing Machinery, New York, NY, USA, v. 51, n. 1, jan 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3143560>.

WATANOBE, Y. et al. Online judge system: Requirements, architecture, and experiences. *International Journal of Software Engineering and Knowledge Engineering*, v. 32, n. 06, p. 917–946, 2022. Disponível em: <https://doi.org/10.1142/S0218194022500346>.

ZHANG, J. Cloud security analysis based on virtualization technology. In: *2022 International Conference on Big Data, Information and Computer Network (BDICN)*. [S.l.: s.n.], 2022. p. 519–522.

APÊNDICES

Apêndice A

Diagramas de robustez

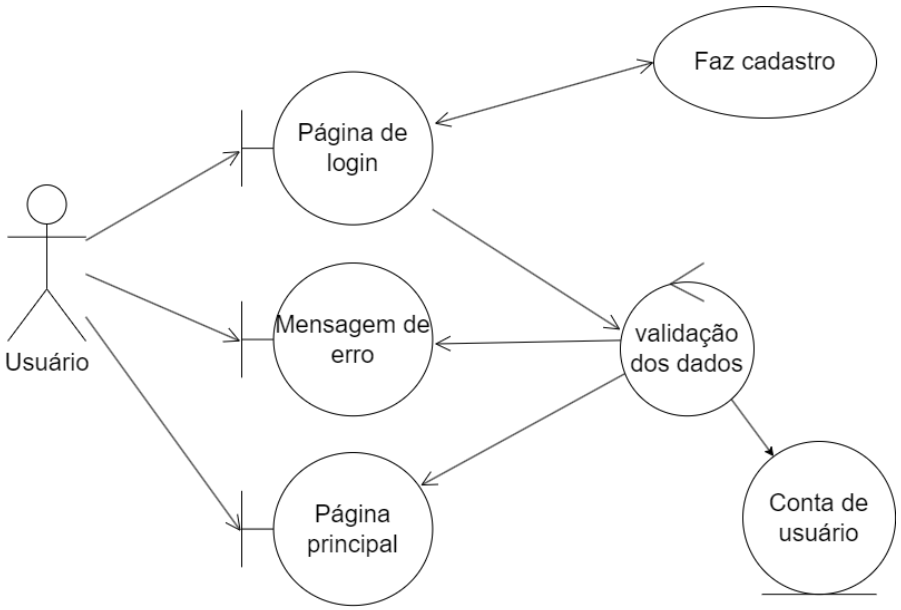

Figura A.1: Diagrama de robustez para RF-001

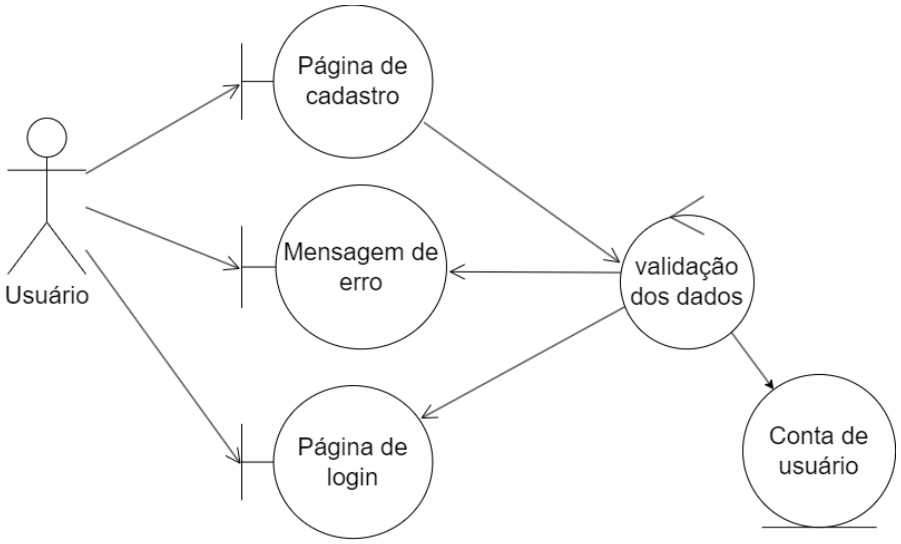

Figura A.2: Diagrama de robustez para RF-002

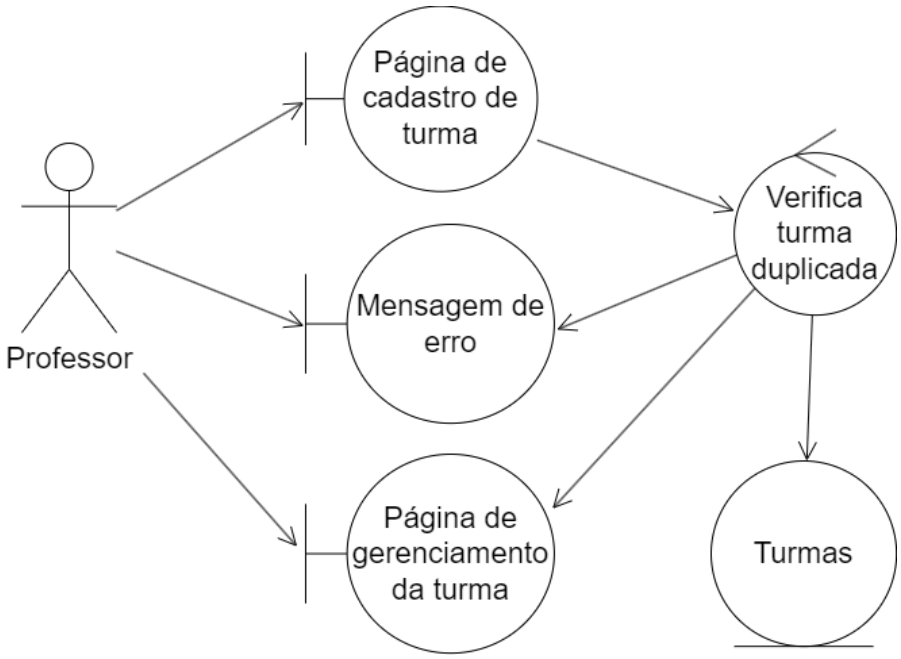

Figura A.3: Diagrama de robustez para RF-003

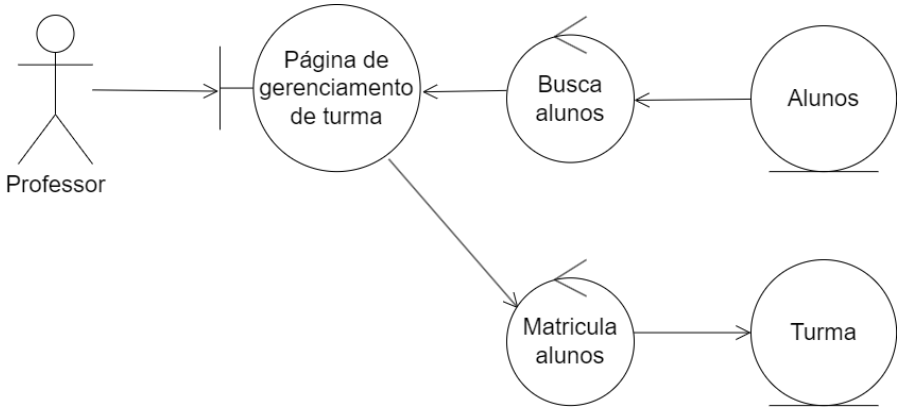

Figura A.4: Diagrama de robustez para RF-004

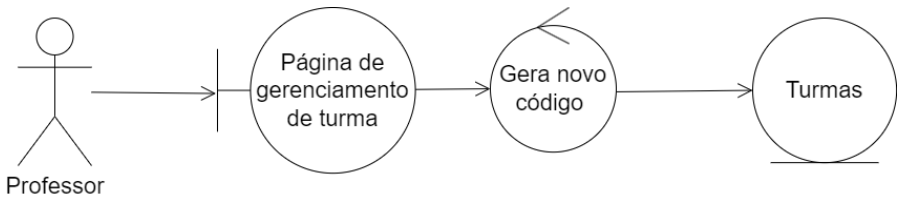

Figura A.5: Diagrama de robustez para RF-005

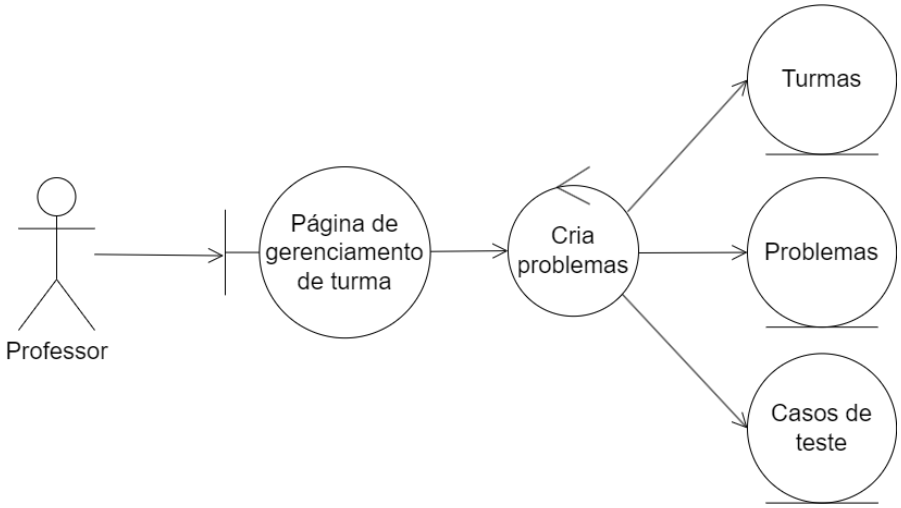

Figura A.6: Diagrama de robustez para RF-006

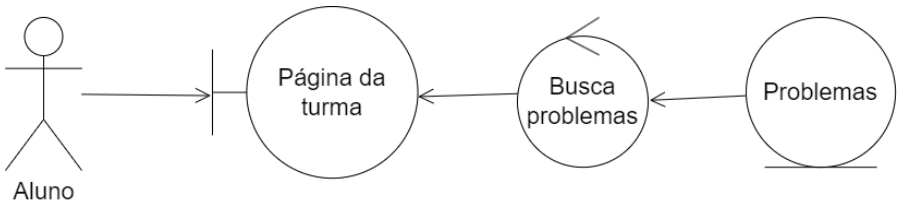

Figura A.7: Diagrama de robustez para RF-007

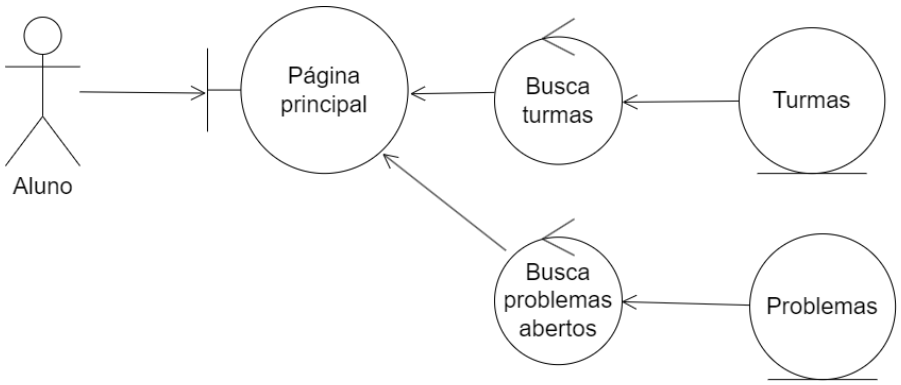

Figura A.8: Diagrama de robustez para RF-008

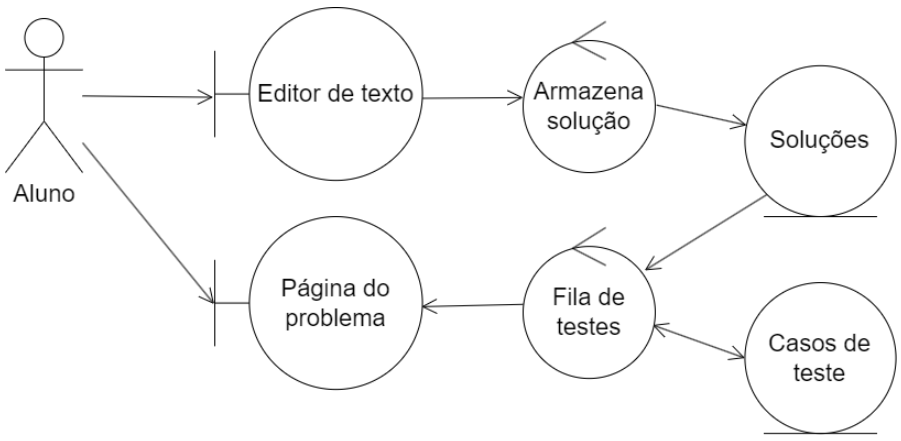

Figura A.9: Diagrama de robustez para RF-009

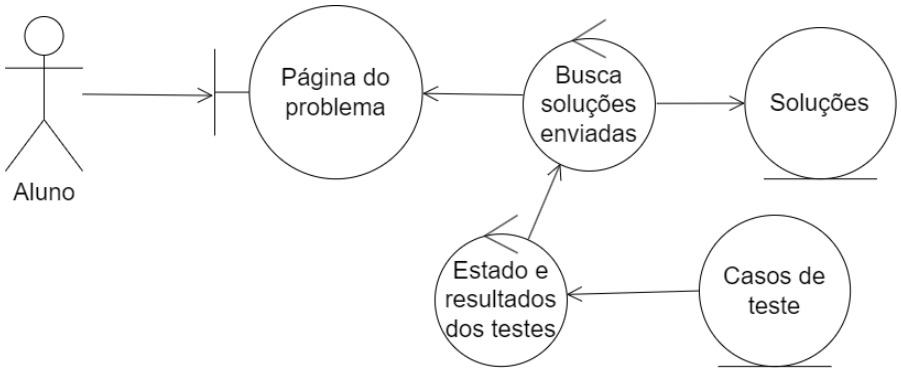

Figura A.10: Diagrama de robustez para RF-010

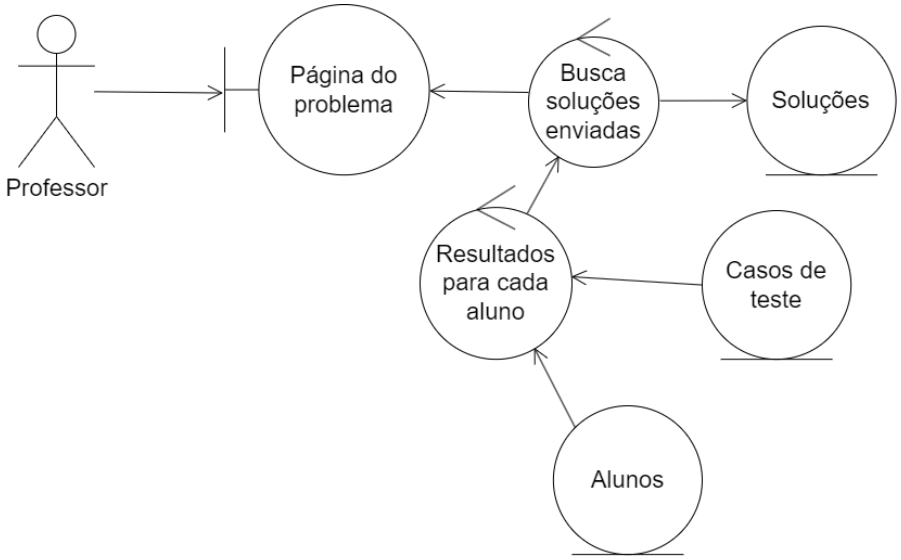

Figura A.11: Diagrama de robustez para RF-011

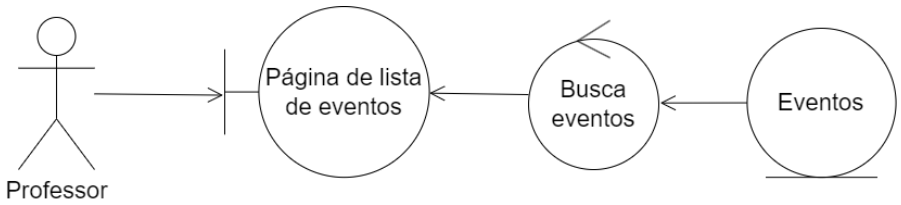

Figura A.12: Diagrama de robustez para RF-012

Apêndice B

Diagramas de sequência

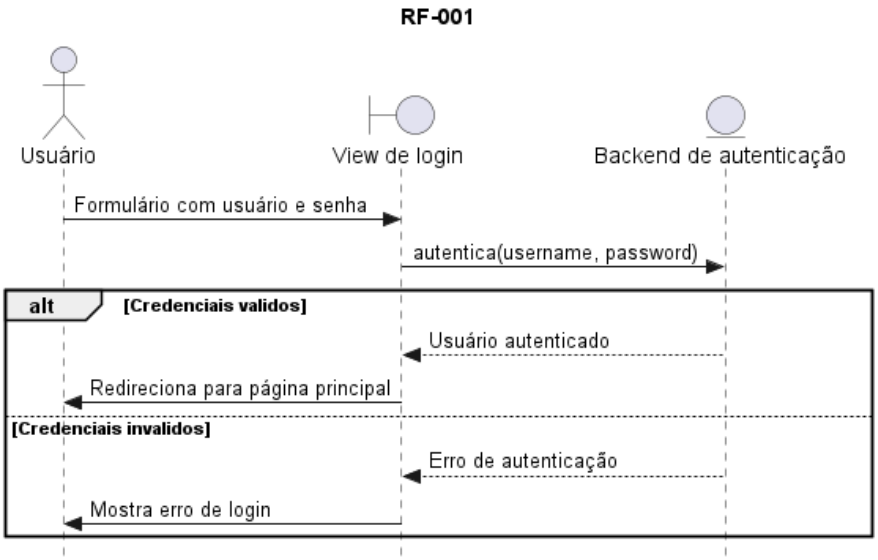

Figura B.1: Diagrama de sequência RF-001

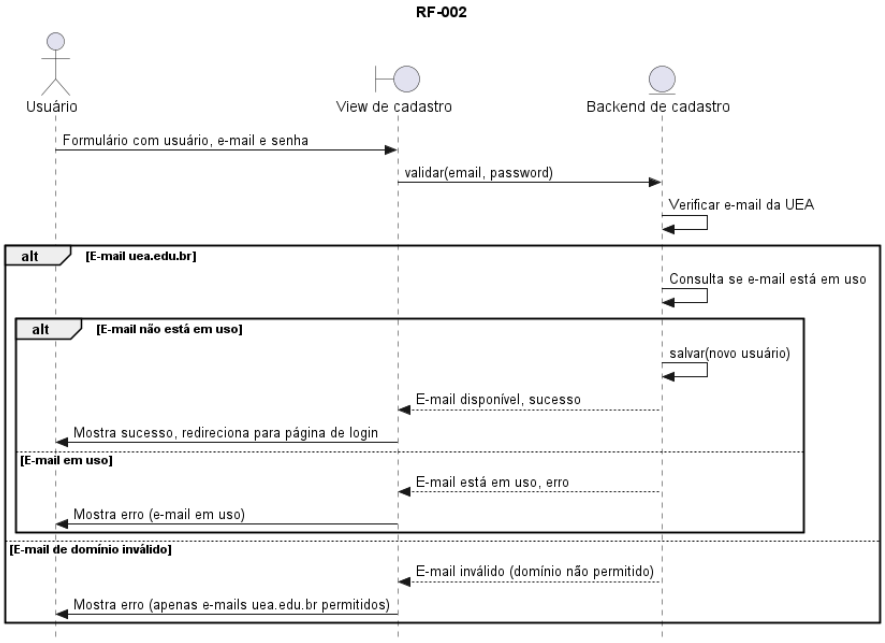

Figura B.2: Diagrama de sequência RF-002

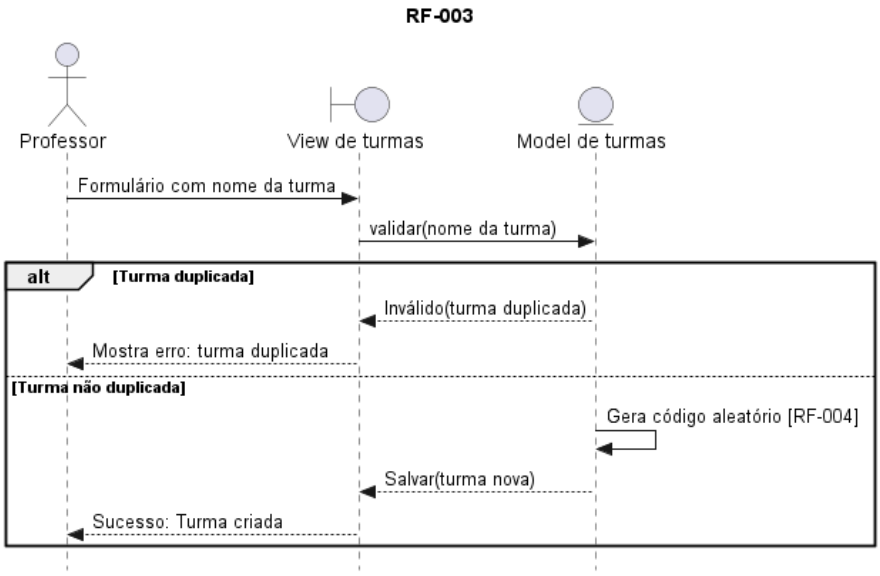

Figura B.3: Diagrama de sequência RF-003

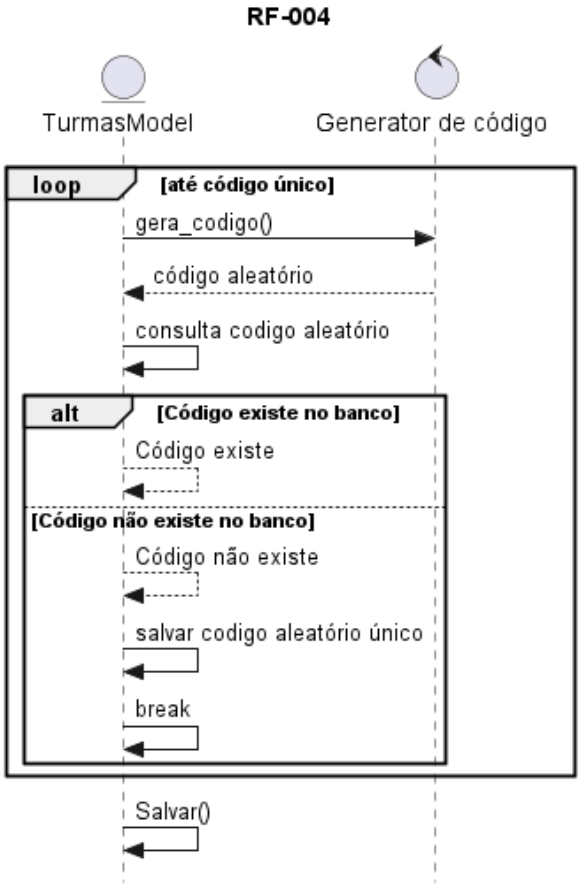

Figura B.4: Diagrama de sequência RF-004

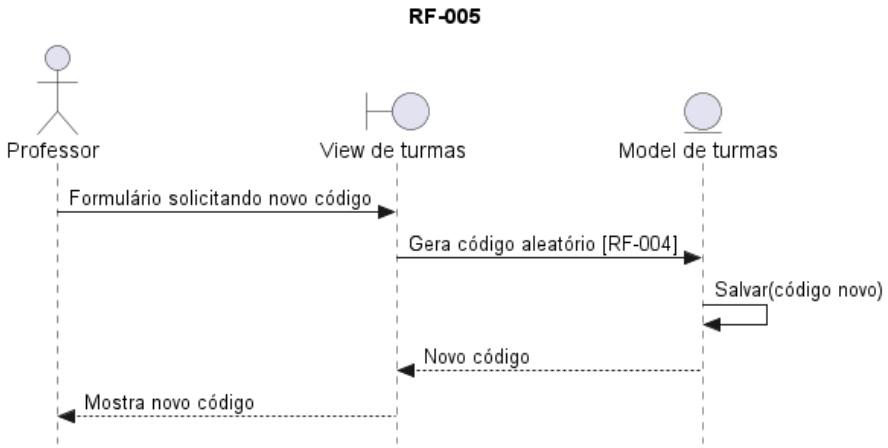

Figura B.5: Diagrama de sequência RF-005

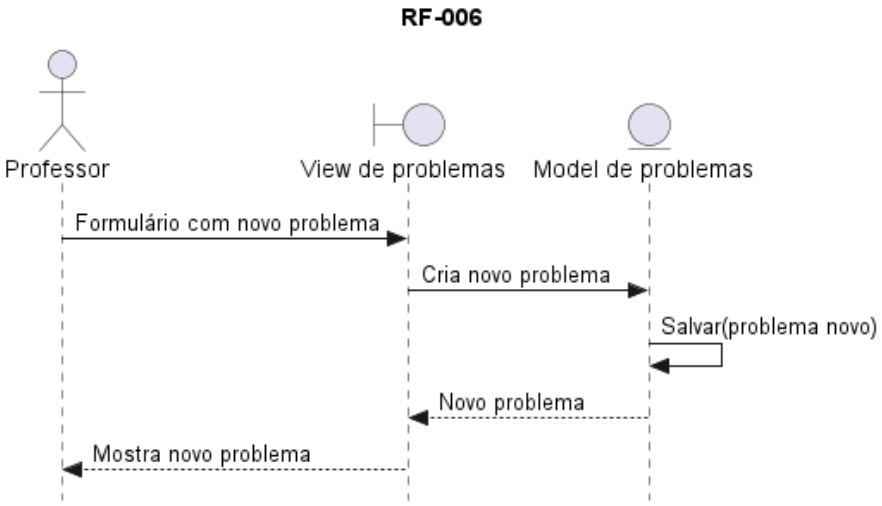

Figura B.6: Diagrama de sequência RF-006

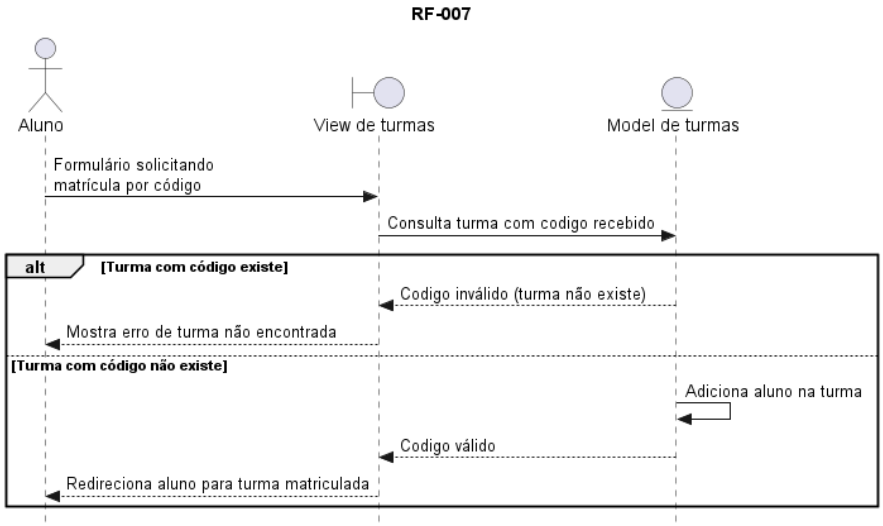

Figura B.7: Diagrama de sequência RF-007

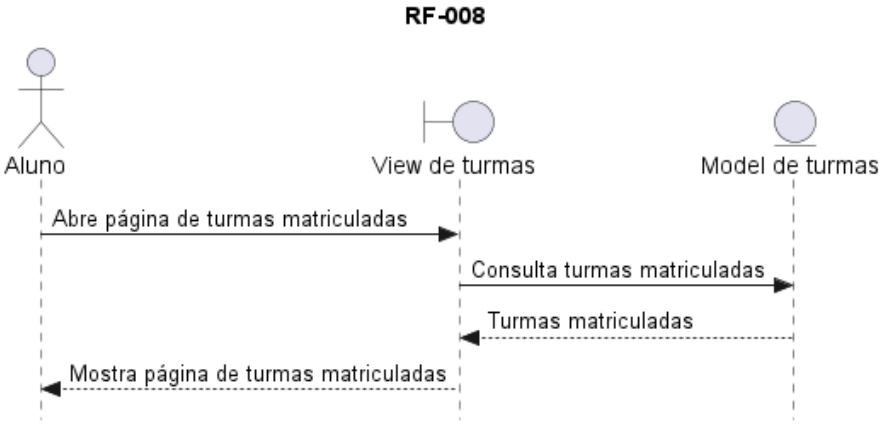

Figura B.8: Diagrama de sequência RF-008

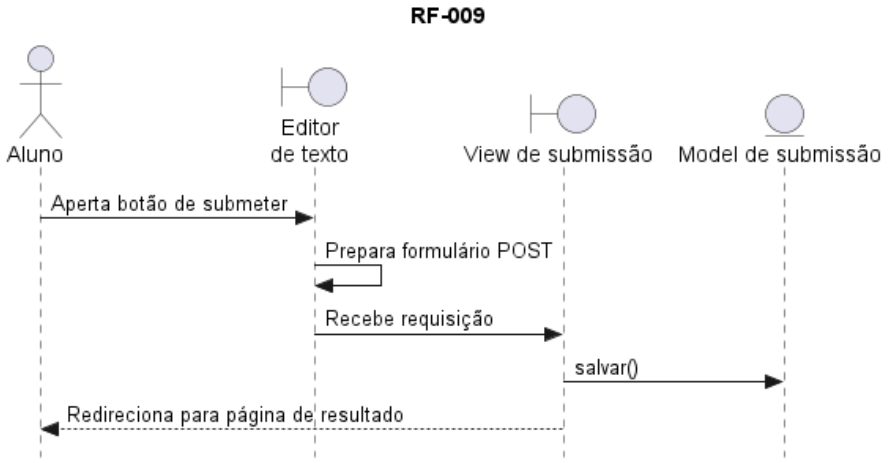

Figura B.9: Diagrama de sequência RF-009

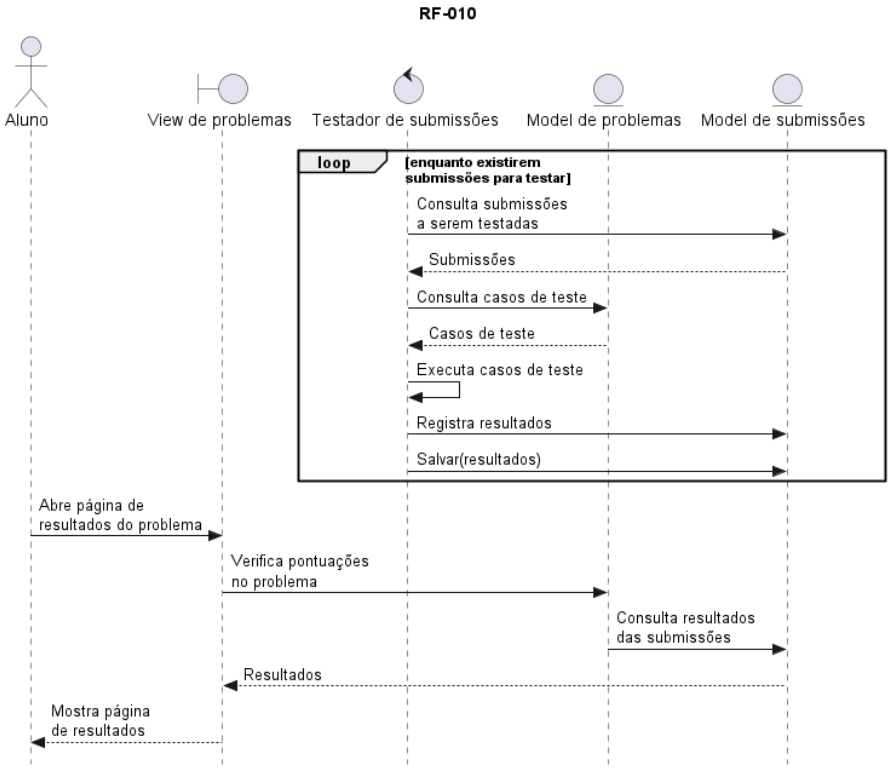

Figura B.10: Diagrama de sequência RF-010

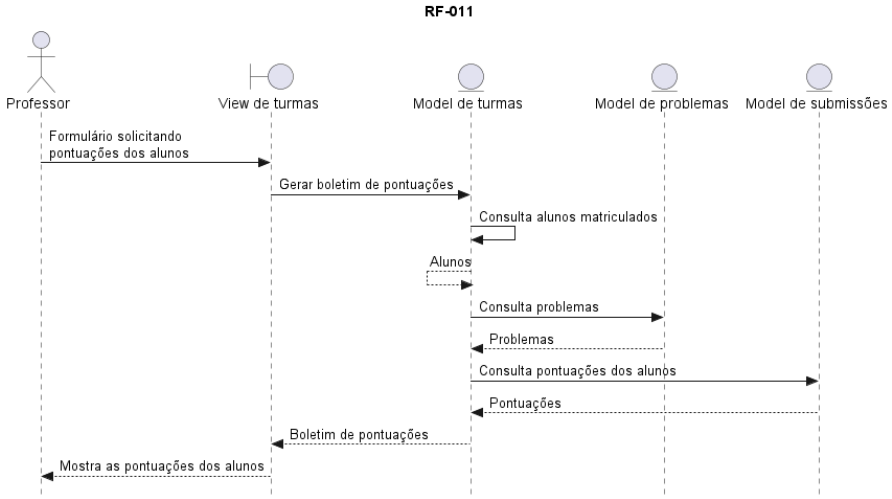

Figura B.11: Diagrama de sequência RF-011

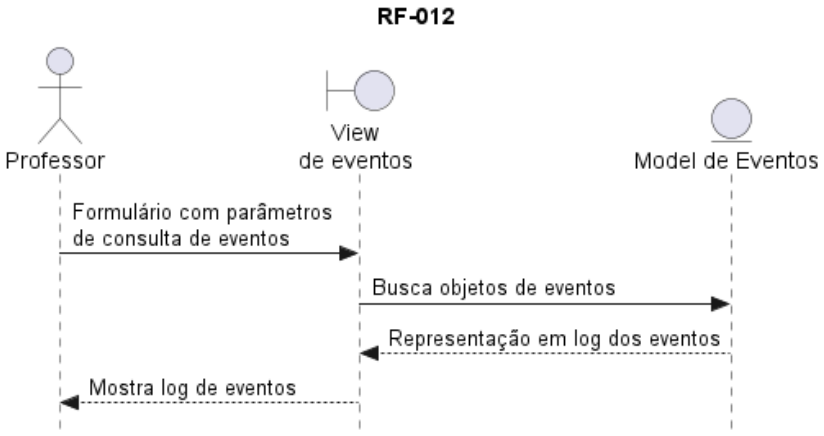

Figura B.12: Diagrama de sequência RF-012