# Documentação

## Projeto de Software

**Professor:** Rohit Gheyi

**Equipe:** Eniedson Fabiano Pereira da Silva Junior;

Eric Diego Matozo Gonçalves;

Francisco Igor Lima Mendes;

Gabriel Mareco Batista de Souto;

Jonathan Lucas Feitosa de Morais;

Lucian Julio Felix da Costa;

Paulo Neto Bezerra de Carvalho;

Pedro Henrique de Oliveira Silva;

Ruan Gomes de Oliveira Alves;

Victor Brandão de Andrade;

Wellisson Gomes Pereira Bezerra Cacho

**Sistema Novo**

## 1. Introdução

### 1.1. Motivação

Na universidade como um todo, independente do curso, muitos alunos encontram dificuldades com as atividades propostas em certas disciplinas e até mesmo com a didática de alguns professores, dificuldades essas que já foram enfrentadas por outros estudantes que já passaram por essas disciplinas. Essas dificuldades em alguns casos podem fazer os estudantes perderem o estímulo para continuar no curso e aumentar a taxa de evasão. A falta de comunicação entre estudantes de diferentes períodos em um mesmo curso, impede que o conhecimento referente a essas atividades que alguns alunos já enfrentaram seja compartilhado para ajudar os novos alunos.

Tendo em vista esse problema, se faz necessária uma aplicação para que os estudantes possam compartilhar suas dúvidas específicas de certas disciplinas com os demais estudantes do curso. Com base nessa necessidade, decidimos desenvolver um website onde os estudantes possam postar suas dúvidas usando tags para indicar a qual disciplina se refere e qual o conteúdo específico que gerou a dúvida, nesse ambiente as dúvidas ficariam disponíveis para que outros estudantes possam ver e responder caso tenham o conhecimento necessário. Esse website, além de ajudar a sanar dúvidas específicas do contexto da universidade, iria estimular a criação de uma comunidade de estudantes mais unida. O website se chama Ask-UFCG, pois ele poderá ser estendido para abranger mais cursos da universidade, mas para o escopo da disciplina de Engenharia de Software vamos nos concentrar apenas no curso de Ciência da Computação.

| | |
|---|---|
| **O problema é...** | Os estudantes enfrentarem dificuldades que seriam facilmente resolvidas com ajuda de alunos que já experienciaram os mesmos problemas. |
| **Que afeta...** | Os estudantes de todos os cursos da universidade. |
| **O impacto disto é...** | Os estudantes dedicarem tempo demais em problemas que poderiam ser facilmente resolvidos e perderem o estímulo de continuar no curso. |
| **A solução seria...** | Um website onde os estudantes poderiam compartilhar suas dúvidas e experiências com outros, promovendo também a criação de uma comunidade de estudantes mais unida. |

## 1.2. Visão da Solução

Auxiliar os estudantes a compartilharem suas dúvidas específicas de certas disciplinas com os demais estudantes, de forma que outros alunos que já passaram pelas mesmas dificuldades quando cursaram as disciplinas possam ajudar.

## 1.3. Visão Geral do Documento

Este documento está organizado em sessões, onde parte delas são referentes ao projeto novo e outras ao projeto real.

- **Projeto Novo:**
    - **Seção 1:** Nesta seção ocorreu a apresentação do problema, a principal motivação para construção do sistema e a visão geral do documento;
    - **Seção 2:** Nesta seção está descrito de forma detalhada a forma como o projeto foi organizado e o planejamento feito para que o desenvolvimento pudesse ocorrer da melhor forma, considerando a gerência do tempo, a gerência dos custos e o gerenciamento dos riscos, bem como os recursos do projeto e a comunicação da equipe;
    - **Seção 3:** Nesta seção é apresentada a definição do processo adotado para o desenvolvimento do projeto, apresentando explicações de

parâmetros que foram usados para fazer a divisão e realização das atividades;
    - **Seção 4:** Nesta seção são descritos os requisitos funcionais e não-funcionais do projeto, incluindo a prototipação das telas do sistema e a descrição dos casos de uso;
- Projeto Real:
    - **Seção 9:** Nesta seção é introduzido o projeto real escolhido pelo grupo e algumas características relacionadas a ele;
    - **Seção 10:** Nesta seção é feita uma análise quanto a qualidade do código do projeto real escolhido utilizando algumas ferramentas de análise estática, como iPlasma, SpotBugs, Checkstyle, entre outras;

## 2. Planejamento

### 2.1. Introdução

A etapa de planejamento é importante em um projeto de software para que seja possível alcançar um produto final de qualidade, dentro do prazo e custo definidos. Essa etapa é essencial, pois é nela que podemos estudar quais os riscos que podem impactar desenvolvimento, como, por exemplo, a falta de experiência do time com certa tecnologia, cronogramas apertados que não permitem que o time tenha o tempo adequado para estudar as tecnologias, além de muitos outros riscos que estão relacionados ao processo de desenvolvimento. Dessa forma, o planejamento é essencial para que seja possível prevenir futuros problemas que possam atrapalhar o desenvolvimento.

Nas primeiras semanas do projeto a equipe reservou tempo para reuniões semanais com o objetivo de definir os requisitos do projeto novo e escolher qual projeto seria utilizado na etapa do projeto real. Nessas reuniões foi discutido não apenas os requisitos do projeto como também as competências de cada um dos membros da equipe para que fosse possível realizar uma alocação eficiente de cada um dos membros. Além disso, foi criado um cronograma de atividades baseado no tempo

disponível para desenvolvimento das tarefas até a entrega. Essa seção do documento descreve como foi feito o planejamento do time.

## 2.2. Gerência do Tempo

A gerência de tempo é uma área muito importante da engenharia de software, pois uma boa gerência de tempo evita a definição de cronogramas apertados que vão desgastar demais a equipe, provavelmente levando a produção de um software sem qualidade e talvez até mesmo à atrasos na entrega do software. As estimativas de tempo iniciais do projeto foram feitas com base no conhecimento empírico dos membros da equipe, considerando também quais os possíveis riscos que seriam enfrentados pelo time.

### 2.2.1. Descrição das Estimativas

Foi levado em consideração que todos os membros da equipe são alunos da graduação, ou seja, estão cursando mais de uma disciplina em paralelo com a disciplina de engenharia de software, algumas das quais podem ter projetos em seus escopos também. Foi importante levar em conta que alguns dos membros trabalham em projetos da universidade por fora da graduação, outros como estagiários e alguns têm empregos por fora do curso. Dessa forma, foi importante considerar o contexto pessoal de cada um dos membros, pois alguém em determinado período pode estar mais sobrecarregado de atividades por conta de tarefas externas ao projeto da disciplina, além de possíveis problemas pessoais e de saúde.

O cronograma inicial do projeto foi construído com a utilização da ferramenta Microsoft Project (MS Project). A seguir está o cronograma produzido para as atividades realizadas até a segunda entrega do projeto.

| | | | |
|---|---|---|---|
| **Sprint 1** | **5,13 dias?** | **Seg 12/07/21** | **Sex 16/07/21** |
| **Projeto Novo** | **5 dias?** | **Seg 12/07/21** | **Sex 16/07/21** |
| Elicitação Requisitos | 2 dias? | Seg 12/07/21 | Ter 13/07/21 |
| Modelagem do Sistema | 3 dias? | Qua 14/07/21 | Sex 16/07/21 |
| Prototipação das Telas | 3 dias? | Qua 14/07/21 | Sex 16/07/21 |
| **Projeto Real** | **5,13 dias?** | **Seg 12/07/21** | **Sex 16/07/21** |
| Configuração do Ambiente e Ferramentas | 5 dias? | Seg 12/07/21 | Sex 16/07/21 |
| Teste das Ferramentas | 5 dias? | Seg 12/07/21 | Sex 16/07/21 |
| Compreensão do Sistema Escolhido | 5,13 dias | Sex 09/07/21 | Sex 16/07/21 |

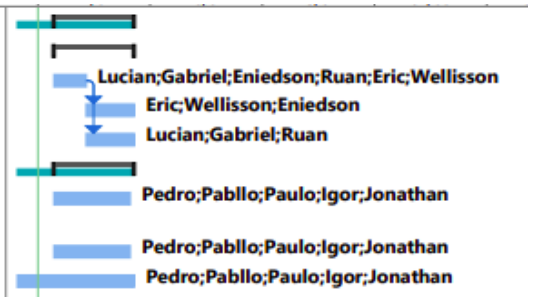

Figura 1 - Cronograma da Sprint 1.

| Sprint 2 | 10,13 dias? | Seg 19/07/21 | Sex 30/07/21 |
|---|---|---|---|
| **Projeto Novo** | **10 dias?** | **Seg 19/07/21** | **Sex 30/07/21** |
| Criação da Base do Projeto | 6 dias? | Seg 19/07/21 | Seg 26/07/21 |
| Estudo Inicial das Tecnologias | 10 dias? | Seg 19/07/21 | Sex 30/07/21 |
| Definição e Alocação das Issues | 4 dias? | Ter 27/07/21 | Sex 30/07/21 |
| **Projeto Real** | **10,13 dias?** | **Seg 19/07/21** | **Sex 30/07/21** |
| Estudo das Métricas do Plugin Metrics | 5 dias? | Seg 19/07/21 | Sex 23/07/21 |
| Identificação de Bad Smells com o Spotbugs | 5 dias? | Seg 19/07/21 | Sex 23/07/21 |
| Execução das Ferramentas de Análise Estáticas (PMD, Spotbugs) | 5 dias? | Seg 19/07/21 | Sex 23/07/21 |
| Análise das Métricas | 5 dias? | Seg 26/07/21 | Sex 30/07/21 |

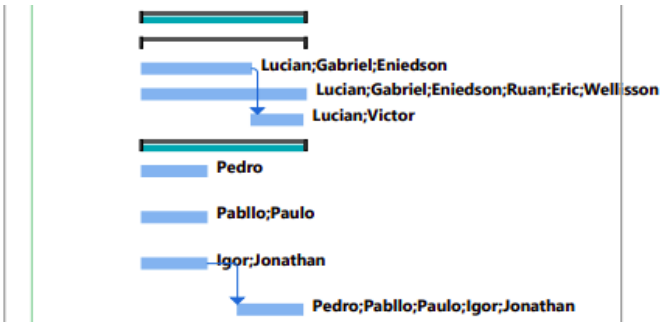


Figura 2 - Cronograma da Sprint 2.

| Sprint 3 | 10,13 dias? | Seg 02/08/21 | Sex 13/08/21 |
|---|---|---|---|
| **Projeto Novo** | **10,13 dias?** | **Seg 02/08/21** | **Sex 13/08/21** |
| Desenvolvimento do Back-end | 10 dias | Seg 02/08/21 | Sex 13/08/21 |
| Desenvolvimento do Front-end | 10 dias? | Seg 02/08/21 | Sex 13/08/21 |
| Teste das Funcionalidades Projeto | 2 dias? | Qui 12/08/21 | Sex 13/08/21 |
| **Projeto Real** | **10,13 dias?** | **Seg 02/08/21** | **Sex 13/08/21** |
| Inicia Documentação dos Resultados | 5 dias? | Seg 02/08/21 | Sex 06/08/21 |
| Reexecução Ferramentas | 5 dias? | Seg 09/08/21 | Sex 13/08/21 |
| Análise dos Novos Resultados | 5 dias? | Seg 09/08/21 | Sex 13/08/21 |

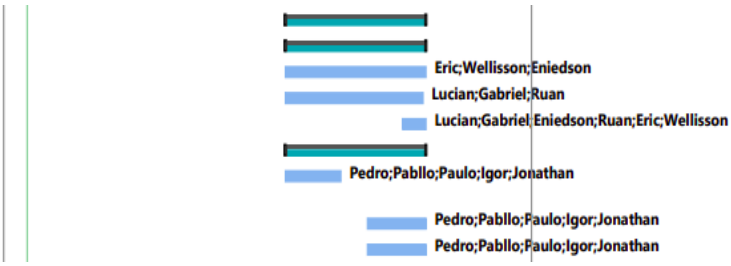


Figura 3 - Cronograma da Sprint 3.

| Sprint 4 | 6,13 dias? | Seg 16/08/21 | Dom 22/08/21 |
|---|---|---|---|
| **Projeto Novo** | **6,13 dias?** | **Seg 16/08/21** | **Dom 22/08/21** |
| Documentação da Entrega | 6 dias? | Seg 16/08/21 | Dom 22/08/21 |
| **Projeto Real** | **6,13 dias?** | **Seg 16/08/21** | **Dom 22/08/21** |
| Finaliza Documentação dos Resultados | 6 dias? | Seg 16/08/21 | Dom 22/08/21 |


Figura 4 - Cronograma da Sprint 4.

Para uma melhor visualização do cronograma segue o link para o pdf do cronograma no repositório com os artefatos relacionados à documentação do projeto.

### 2.2.2. Análise das Estimativas

Nesta seção vamos analisar o que foi definido durante a produção do cronograma com relação ao tempo real gasto nas atividades do projeto.

| Atividade | Estimativa (em dias) | Duração (em dias) |
|---|---|---|
| **Projeto Novo** | | |
| Elicitação dos Requisitos | 2 | 2 |
| Modelagem do Sistema | 3 | 3 |
| Prototipação das Telas | 3 | 3 |
| **Projeto Real** | | |
| Configuração do Sistema e das Ferramentas | 5 | 5 |
| Testes das Ferramentas | 5 | 5 |
| Compreensão do Sistema | 5 | 5 |

Estimativas de tempo da primeira sprint.

| Atividade | Estimativa (em dias) | Duração (em dias) |
|---|---|---|
| **Projeto Novo** | | |
| Criação da Base do Projeto | 6 | 3 |
| Estudo Inicial das Tecnologias | 10 | 10 |
| Definição e Alocação das Issues | 4 | 4 |
| **Projeto Real** | | |
| Estudo das Métricas do Plugin Metrics | 5 | 5 |
| Identificação de Bad Smells com Spotbugs | 5 | 5 |
| Execução das Ferramentas de Análise Estática | 5 | 5 |
| Análise das Métricas | 5 | 5 |

Estimativas de tempo da segunda sprint.

| Atividade | Estimativa (em dias) | Duração (em dias) |
|---|---|---|
| **Projeto Novo** | | |
| Desenvolvimento do Back-end | 10 | 10 |
| Desenvolvimento do Front-end | 10 | 10 |
| Teste das Funcionalidades | 2 | 2 |
| **Projeto Real** | | |
| Inicia Documentação dos Resultados | 5 | 7 |
| Reexecução das Ferramentas | 5 | 3 |
| Análise dos Novos Resultados | 5 | 6 |

Estimativas de tempo da terceira sprint.

| Atividade | Estimativa (em dias) | Duração (em dias) |
|---|---|---|
| **Projeto Novo** | | |
| Documentação da Entrega | 6 | 8 |
| **Projeto Real** | | |
| Finaliza Documentação dos Resultados | 6 | 8 |

Estimativas de tempo da quarta sprint.

## 2.3. Gerência do Custo

O cálculo do custo feito pela equipe levou em consideração os gastos com pessoal e com os equipamentos necessários para o desenvolvimento do projeto. Os valores estão em moeda corrente nacional (R$).

- Custo:
  - Diversos:
    - 200 (água, luz, aluguel) * 12 membros * 4 meses = R$9.600,00
  - Salário dos membros (11 desenvolvedores e 1 gerente):
    - ((1500 * 11) + 2500) * 4 meses = R$76.000,00
  - Equipamentos:
    - Devido a pandemia, não foi possível utilizar os computadores da universidade, então todos os membros utilizaram seus computadores pessoais, ou seja, não houve custo adicional com ferramentas.

| **Tipo de Custo** | **Custo (R$)** |
|---|---|
| Pessoal (salário) | R$76.000,00 |
| Equipamentos | R$0,00 |
| Diversos | R$9.600,00 |
| Erro | R$17.120,00 |
| Lucro (30% em cima do erro) | R$30.816,60 |
| **Total** | R$133.536,00 |

## 2.4. Gerência de Riscos

Visto que alguns problemas são recorrentes na área de desenvolvimento de software, como, por exemplo, a curva de aprendizado de alguma tecnologia específica solicitada em determinado projeto. A atividade de gerência de riscos é uma das áreas mais importantes da engenharia de software, pois realizar uma boa gerência de riscos

ajuda a identificar possíveis problemas e planejar maneiras de minimizar esses problemas ou até mesmo evitá-los.

| Classificação do Risco | Impacto e Descrição do Risco | Estratégia de Diminuição e/ou Plano de Contingência |
|---|---|---|
| Alto | Definição de um sistema com muitos requisitos. | Dividir os requisitos pensados em essenciais e extras. De forma que os extras só serão implementados dentro do escopo da disciplina caso haja tempo. |
| Alto | Escolha de um projeto complicado de executar as ferramentas para o projeto real. | Escolher um projeto com suporte ao maven e pudesse ser compilado facilmente. |
| Alto | Dificuldade com as ferramentas do projeto real. | Iniciar o estudo e execução das ferramentas o mais rápido possível para poder identificar e corrigir problemas o mais cedo possível. |
| Alto | Curva de aprendizado das tecnologias do projeto novo. | Iniciar o estudo das tecnologias o mais rápido possível e solicitar ajuda a membros mais experientes do time. |
| Alto | Os membros do time ficarem indisponíveis por conta de atividades da graduação, projeto ou pessoais. | Definir um cronograma onde a margem do tempo de desenvolvimento seja razoável, de forma a não gerar atrasos. |

| | | |
|---|---|---|
| Alto | Pandemia da Covid-19 é um risco alto, pois o fato dos membros não poderem se encontrar pode causar falhas de comunicação importantes. | Para contornar isso, foram definidas reuniões semanais e criado um servidor no discord para que todo o time pudesse se comunicar da melhor forma possível. |

Tabela 1. Riscos identificados.

## 2.5. Recursos do Projeto

a) Hardware

Devido a questão da pandemia do Covid-19, os membros da equipe não tiveram a possibilidade de se reunir nos laboratórios da universidade. Dessa forma todos ficaram restritos aos seus computadores pessoais.

b) Pessoas

i) Eniedson Fabiano Pereira da Silva Junior: Projeto Novo (Back-end)
ii) Eric Diego Matozo Gonçalves: Projeto Novo (Back-end)
iii) Francisco Igor Lima Mendes: Projeto Real
iv) Gabriel Mareco Batista de Souto: Projeto Novo (Front-end)
v) Jonathan Lucas Feitosa de Morais: Projeto Real
vi) Lucian Julio Felix da Costa: Projeto Novo (Front-end)
vii) Paulo Neto Bezerra de Carvalho: Projeto Real
viii) Pedro Henrique de Oliveira Silva: Projeto Real
ix) Ruan Gomes de Oliveira Alves: Projeto Novo (Front-end)
x) Victor Brandão de Andrade: Gerente do Projeto
xi) Wellisson Gomes Pereira Bezerra Cacho: Projeto Novo (Back-end)

c) Software

1. GitHub: Foi a plataforma escolhida para fazer o versionamento do código;
2. Heroku: Foi a plataforma escolhida para fazer o deploy da aplicação;
3. Google Docs: Ferramenta escolhida para a produção de toda a documentação do projeto;
4. Discord: Plataforma escolhida para realizar toda a comunicação entre os membros da equipe e realizar as reuniões;
5. Figma: Ferramenta escolhida para realizar a prototipação das telas do projeto;
6. LucidChart: Ferramenta utilizada para criar os diagramas;
7. Visual Studio Code (VSCode): Editor de texto escolhido para desenvolver o front-end da aplicação;
8. IntelliJ: IDE escolhida para o desenvolvimento do back-end da aplicação;
9. ReactJS: Framework escolhido para desenvolvimento do front-end;
10. Ant-Design: Biblioteca de componentes utilizada no desenvolvimento do front-end;
11. SpringBoot: Framework Java utilizado para desenvolvimento do back-end;

## 2.6. Comunicação

A equipe de desenvolvimento utilizou o discord como principal canal de comunicação, foi criado um servidor no discord onde foram criados diversos canais de texto separados para os projetos novo e real, além de canais mais gerais que englobam toda a equipe, como, por exemplo, um canal para dúvidas e um para salvar links de materiais interessantes sobre as ferramentas utilizadas no projeto. O discord também foi utilizado para as reuniões e para a aplicação da técnica de pair programming através do recurso de compartilhamento de tela, prática que foi importante para casos onde um dos membros precisava de ajuda dos membros mais experientes da equipe.

Foi definido que teríamos duas reuniões por semana, sendo uma delas geral, com todos os membros e durante o horário da aula de engenharia de software de segunda-feira. A segunda reunião semanal foi dividida, os times do projeto real e novo

tinham suas reuniões separadamente e em paralelo durante o horário da aula de engenharia de software na quinta-feira. Outras reuniões que se mostraram necessárias foram marcadas em horários por fora desses, em que todos os membros que precisavam estar presentes poderiam comparecer.

## 3. Processo

Para que o gerenciamento dos recursos disponíveis para o projeto fosse feito da melhor forma possível, decidimos escolher a metodologia ágil Scrum como a melhor alternativa para organizar a equipe. Dessa forma, seguindo a metodologia, as nossas atividades foram separadas em sprints bem definidas de forma que ao final de todas as sprints houvesse algum incremento no produto sendo desenvolvido (o software e a documentação do projeto). Além disso, foram definidas reuniões semanais, ao menos duas vezes por semana para discutir os avanços e possíveis impedimentos do time, de forma que qualquer problema pudesse ser rapidamente identificado e resolvido.

Os membros do time foram divididos em duas frentes, projeto real com 4 membros e o projeto novo com 6 membros, sendo 3 para o front-end e 3 para o back-end. Para facilitar a comunicação entre os membros foi criado um um sistema hierárquico, onde foi definido um gerente geral para o projeto e dois sub-líderes, sendo um para o projeto real e outro para o projeto novo. A partir dessa divisão as reuniões semanais ficaram divididas de forma que uma era uma reunião geral com todo o time e a outra reunião era de cada equipe em separado (projeto real e novo) e de maneira paralela. Além disso, foi decidido manter atas apenas para reuniões em que decisões fossem tomadas, como forma de documentar as decisões, ou seja, não havia atas para reuniões de status.

Para manter um histórico das funcionalidades e mantê-las bem documentadas, foi utilizado um sistema de issues nos repositórios do projeto novo no GitHub, onde para cada funcionalidade a ser desenvolvida havia uma issue documentando o que deveria ser feito e a alocação de um dos membros do projeto. Para manter a qualidade do código produzido no projeto novo, foi definido que sempre que uma funcionalidade fosse finalizada e um PR (Pull Request) fosse aberto, ao menos dois membros do time deveriam revisar o código e solicitar mudanças caso necessário.

Para simplificar o processo, foi decidido que a documentação de código só seria utilizada em funções mais complexas, onde apenas o nome da função não facilita o seu entendimento. Além disso, outra simplificação foi a automatização do deploy do back-end da aplicação para o Heroku no repositório do GitHub, de forma que sempre que a branch main seja atualizada, o deploy seja feito automaticamente, evitando futuras complicações na etapa de disponibilizar a aplicação para os usuários.

## 4. Requisitos

### 4.1. Requisitos Não-Funcionais

| Cód. | Nome | Prioridade |
|---|---|---|
| RNF-01 | A busca das perguntas deve ser finalizada em até 5 segundos. | Essencial |
| RNF-02 | Renderizar a página inicial em média em 5 segundos. | Essencial |
| RNF-03 | O sistema deve executar minimamente bem em qualquer navegador. | Essencial |

### 4.2. Requisitos Funcionais

| Cód. | Nome | Prioridade |
|---|---|---|
| RF-01 | Login de usuário | Essencial |
| RF-02 | Recuperar as perguntas de um usuário | Essencial |
| RF-03 | Fazer uma pergunta | Essencial |
| RF-04 | Editar uma pergunta | Essencial |
| RF-05 | Deletar uma pergunta | Essencial |

| | | |
|---|---|---|
| RF-06 | Adicionar um comentário a uma resposta | Essencial |
| RF-07 | Editar um comentário em uma resposta | Essencial |
| RF-08 | Recuperar todos os comentários de uma resposta | Essencial |
| RF-09 | Apagar um comentário em uma resposta | Essencial |
| RF-10 | Responder uma pergunta | Essencial |
| RF-11 | Editar uma resposta | Essencial |
| RF-12 | Deletar uma resposta | Essencial |
| RF-13 | Cadastrar um usuário | Essencial |
| RF-14 | Editar um usuário | Essencial |
| RF-15 | Buscar perguntas por filtro | Essencial |
| RF-16 | Buscar perguntas por tag | Essencial |
| RF-17 | Buscar perguntas por título | Essencial |

### 4.2.1 Especificação dos Requisitos

A especificação completa dos requisitos funcionais pode ser visualizada neste link.

| **RF-01** | | | |
|---|---|---|---|
| **Nome:** | Login de usuário | | |
| **Descrição:** | O sistema deve permitir que o usuário logue em sua conta ask-ufcg, utilizando seu endereço de email e senha. | | |
| **Atores:** | Usuário | | |
| **Prioridade:** | Essencial | **Anexo:** | Nenhum. |
| **Requisitos Não Funcionais Associados:** | Nenhum. | | |
| **Entradas e pré-condições:** | O usuário já está cadastrado. | | |

| | |
|---|---|
| **Saídas e pós-condições:** | O usuário é levado a tela de Home. |
| **Fluxos de eventos** | |
| **Fluxo principal:** | 1. O usuário clica no botão Entrar no canto superior direito.<br>2. O usuário preenche o formulário com seu e-mail e sua senha.<br>3. O sistema valida os dados fornecidos pelo usuário.<br>4. O usuário clica no botão vermelho Entrar.<br>5. O sistema checa se o usuário já está cadastrado.<br>6. O usuário é direcionado para sua tela de Home. |
| **Fluxo secundário 1:** | Se no passo 3 do fluxo principal o sistema constatar que o usuário forneceu dados inválidos, o usuário retornará ao passo 1. |
| **Fluxo secundário 2:** | Se no passo 5 o sistema constatar que o usuário ainda não foi cadastrado, o usuário será redirecionado para a tela de cadastro. |

| | | | |
|---|---|---|---|
| **RF-03** | | | |
| **Nome:** | Fazer pergunta | | |
| **Descrição:** | O sistema deve permitir que um usuário possa criar uma pergunta. | | |
| **Atores:** | Usuário | | |
| **Prioridade:** | Essencial | **Anexo:** | Nenhum. |
| **Requisitos Não Funcionais Associados:** | Nenhum. | | |
| **Entradas e pré-condições:** | O usuário estar logado. | | |

| | |
|---|---|
| **Saídas e pós-condições:** | A pergunta do usuário será registrada no sistema e estará disponível para todos os usuários verem. |
| **Fluxos de eventos** | |
| **Fluxo principal:** | 1. O usuário clica no botão Faça uma pergunta no canto superior direito da tela Home.<br>2. O usuário é redirecionado para a tela de criação de pergunta.<br>3. O usuário preenche o formulário com a categoria, o título e a descrição da pergunta.<br>4. O usuário clica no botão cadastrar nova pergunta.<br>5. O sistema valida os dados fornecidos pelo usuário.<br>6. O usuário é redirecionado para a tela Home e lá estará disposta a sua pergunta, junto com as que foram previamente também feitas. |
| **Fluxo secundário 1:** | Se no passo 5 do fluxo principal o sistema constatar que o usuário forneceu dados inválidos, retorna-se ao passo 3. |

| **RF-13** | | | |
|---|---|---|---|
| **Nome:** | Cadastrar um usuário. | | |
| **Descrição:** | O sistema deve permitir o cadastro de um usuário. | | |
| **Atores:** | Usuário | | |
| **Prioridade:** | Essencial | **Anexo:** | Nenhum. |
| **Requisitos Não Funcionais Associados:** | Nenhum. | | |

| | |
|---|---|
| **Entradas e pré-condições:** | Nenhum. |
| **Saídas e pós-condições:** | O usuário ser cadastrado no sistema. |
| **Fluxos de eventos** | |
| **Fluxo principal:** | 1. Na tela de Cadastro, o usuário preenche os seus dados.<br>2. O usuário clica no botão cadastrar.<br>3. O sistema verifica se os dados passados pelo usuário são válidos.<br>4. O usuário é cadastrado no sistema.<br>5. O usuário é redirecionado para a tela de login. |
| **Fluxo secundário 1:** | Se no passo 3 do fluxo principal o sistema constatar que o usuário forneceu dados inválidos, retorna-se ao passo 1. |

| | | | |
|---|---|---|---|
| **RF-15** | | | |
| **Nome:** | Buscar perguntas por filtro. | | |
| **Descrição:** | O sistema deve permitir que as perguntas sejam pesquisadas baseadas em filtros. Os filtros são: new (perguntas mais recentes), vote (as perguntas com mais likes), relevant (as perguntas mais relevantes com base na diferença entre likes e dislikes) e answered (as perguntas que já foram respondidas). | | |
| **Atores:** | Usuário | | |
| **Prioridade:** | Essencial | **Anexo:** | Nenhum. |
| **Requisitos Não Funcionais Associados:** | RNF-01 | | |
| **Entradas e pré-condições:** | O usuário indicar os filtros. | | |
| **Saídas e pós-condições:** | As perguntas filtradas são exibidas na tela. | | |
| **Fluxos de eventos** | | | |

| | |
|---|---|
| **Fluxo principal:** | 1. O usuário na tela home seleciona o filtro da busca.<br>2. O usuário clica no botão de realizar busca.<br>3. O sistema faz a filtragem das perguntas com base no filtro selecionado.<br>4. O sistema exibe as perguntas filtradas para o usuário. |
| **Fluxo secundário 1:** | Se no passo 3 do fluxo principal o sistema não encontrar nenhum pergunta, o sistema irá mostrar uma mensagem avisando isso ao usuário. |

## 4.3. Diagrama de Casos de Uso

O diagrama de casos de uso pode ser visualizado no link a seguir:

- Diagrama

## 4.4. Prototipação de Telas

A prototipação das telas pode ser vista no link a seguir:

- Telas

## 5. Projeto Arquitetural

## 6. Especificação Formal

## 7. Implementação

## 8. Testes

**Sistema Real**

## 9. Introdução

O projeto tratado nas seções 10, 11 e 12 é chamado “Web Magic”. Ele é um “web crawler” cujo objetivo é simplificar esse papel, sendo customizável para uma aplicação específica. Ele fornece funcionalidades como baixar, gerenciar URL, extrair o conteúdo e persistência de dados. Além de dar suporte a uma API simples e capacidade de “multi-threading”.

## 10. Qualidade

### 10.1 Análise:

#### 10.1.1 Metrics:

Nesta seção, descreveremos todas as métricas, coletadas a partir do plugin Metrics, e seus impactos na qualidade do projeto. Abaixo, é possível ver a tabela contendo os resultados obtidos:

| Métrica | Resultado |
|---|---|
| Atribute Hiding Factor (AHF) | 89,20% |
| Attribute Inherritance Frequency (AIF) | 13,55% |
| Number of Classes (C) | 313 |
| Coupling Factor (CF) | 3,51% |
| Lines of Java (L(J)) | 15.522 |
| Number of Leaf Classes | 264 |
| Method Hiding Factor (MHF) | 20,96% |
| Method Inheritance Factor (MIF) | 20,97% |
| Polymorphism Factor (PF) | 57,39% |
| True Comment Ratio (TCOM_RAT) | 14,29% |
| Average Cyclomatic Complexity (v(G)avg) | 1,71 |
| Total Cyclomatic Complexity (v(G)tot) | 1.718 |
| Number of Test Classes (Ct) | 99 |

| | |
|---|---|
| Javadoc Class Coverage (Jc) | 98,72% |
| Javadoc Method Coverage (Jm) | 15,00% |

**Tabela X:** Métricas de projeto executadas no projeto real e seus respectivos resultados.

A partir das métricas obtidas na Tabela X, estão abaixo suas descrições e análises.

**Attribute Hiding Factor (AHF):** indica uma média do quanto um atributo é visto para outras classes fora sua própria classe. Por este atributo apresentar um valor alto, cerca de 90%, idealmente esta métrica deveria estar o mais próximo possível de 100%, que indicaria que todos os atributos estão ocultos.

**Attribute Inherritance Frequency (AIF):** informa qual a taxa de atributos que estão disponíveis graças à herança na média das classes. O resultado obtido foi de 13,55%, idealmente o valor deveria ser 0%, o que indicaria que não haveria herança alguma de atributos (todos os atributos declarados como privados).

**Number of Classes (C):** indica o número de classes (que não são anônimas internas) presentes no projeto. Na análise foram identificadas 313 classes, este número pode ser considerado razoável pela complexidade do projeto por seu número de linhas.

**Lines of Java (L(J)):** diz respeito ao número de linhas em Java que o projeto possui. Seu resultado é de 15522 linhas, indicando que o projeto é intermediariamente complexo.

**Number of Leaf Classes:** indica o número de classes folha (classes que não possuem subclasses). O resultado obtido foi 264, sendo assim, podemos assumir que boa parte das classes tem propósito pontual e apresentam a borda do projeto, visto que 264 classes do total de 313 são classes folha.

**Method Hiding Factor (MHF):** trata da proporção da média de métodos que são visíveis a outras classes. Seu valor é de 20,96%, este valor está dentro do faixa de referência, entre 8% e 25%. Isto aponta um bom equilíbrio entre o número de funcionalidades e uso de implementações de abstrações.

**Method Inheritance Factor (MIF):** diz respeito à proporção  entre a média de métodos herdados por outras classes, ao invés da própria classe. Seu resultado foi semelhante à métrica “Method Hiding Factor“, com o valor de 20,97%. Está dentro da faixa de referência, que seria é entre 20% e 80%, porém ainda apresenta um baixo número abstrações a partir de herança. Ademais, é possível inferir que há uma relação entre as métricas MHF e MIF, isto nos leva a concluir que métodos geralmente visíveis às outras classes também são herdados.

**Polymorphism Factor (PF):** indica a probabilidade de uma classe ser sobrescrita por uma subclasse, conhecida como fator de polimorfismo. Seu valor é de 57,39%, o que indica uma taxa de polimorfismo muito alta. Este valor foi obtido graças ao uso de excessivo de *override* ao longo do código. Apesar de valores até 10% de PF trazerem benefícios (como a redução na densidade de defeitos), valores muito acima dos 10% diminuem esses benefícios.

**True Comment Ratio (TCOM_RAT):** indica a proporção de número de comentários por código. Por esta taxa apresentar o valor de 14,29%, o que apresenta

um bom número comentários dado o tamanho do projeto.

**Average Cyclomatic Complexity (v(G)avg):** apresenta o valor médio da complexidade ciclomática de todos os métodos não abstratos. A complexidade ciclomática de um método é definido como a quantidade de caminhos não lineares presentes neste. Caminhos não lineares são entendidos como fluxos alternativos de um mesmo código, em que há alguma diferença no fluxo de execução. Por exemplo, a presença de condicionais if-else aumentam a complexidade ciclomática de um método se comparado com um código sem verificações condicionais.

O valor obtido desta métrica foi de 1,71 na média, geralmente quanto menor o valor, menos ciclomaticamente complexo o projeto é. Este valor está na faixa de risco mínimo, entre 1 e 10, isto indica que a maior parte dos métodos são relativamente simples e apresentam um pequeno número de fluxos alternativos. Sendo assim, há um baixo risco do projeto não cumprir uma tarefa devido a defeitos no fluxo.

**Total Cyclomatic Complexity (v(G)tot):** é semelhante à “v(G)avg”, porém diz respeito à complexidade ciclomática total e não média. Seu valor obtido foi de 1718, indicando que ao longo de todo o projeto há cerca de 1718 fluxos diferentes a serem executados. Apesar deste número de fluxos diferentes, o v(G)avg é baixo, então isto não apresenta um risco para o projeto.

**Number of Test Classes (Ct):** diz respeito à quantidade de classes de teste. Seu valor foi de 99 classes, o que significa que cerca de apenas 50% das classes foram testadas (visto que o projeto possui um total de 313 classes e destas 99 são de teste). Isto indica um número aceitável, porém abaixo do esperado no número de classes de testes ao longo do projeto.

**Javadoc Class Coverage (Jc):** indica a porcentagem de classes que possuem javadoc. O resultado obtido foi de 98,72%, logo, o projeto é bem documentado no que diz respeito às classes.

**Javadoc Method Coverage (Jm):** indica a porcentagem de métodos que possuem javadoc. Seu valor foi de 15,00%, portanto o número de classes documentadas é satisfatório. Esta métrica tem uma relação com a TCOM_RAT, que foi de 14,29%. Esta relação reforça que a documentação javadoc, no que diz respeito aos métodos, é satisfatória dado o tamanho do projeto.

A partir da análise das métricas descritas e registradas acima é possível assumir que: o projeto é de baixo a intermediariamente complexo pelo número de classes de linhas e pela complexidade ciclomática; possui um número exagerado de *override*; o padrão especialista da informação é afetado por possuir atributos visíveis às outras classes graças a herança; o fluxo de execução é conciso; a documentação javadoc é satisfatória nas classes e métodos; o projeto apresenta um valor aceitável de testes, porém um pouco abaixo do esperado.

### 10.1.2 Bad smells:

Nesta seção foi tratada a análise estática referente às más práticas de código presentes, através da ferramenta “SpotBugs”. Na figura X foram identificados os bugs detectados pela ferramenta e sua descrição.


**Figura X.** Indicação de todos os bugs identificados no projeto.

Foram identificados um total de 58 tipos de bug com a ferramenta SpotBugs, separados em 7 categorias, conforme identificado na tabela X. Na categoria esse 10.1.2.1 é feita uma análise mais minuciosa sobre estes bugs.

| Categoria de Bug | Quantidade identificada |
|---|---|
| **Dodgy code** | 9 |
| **Performance** | 10 |
| **Bad practice** | 7 |

| | |
|---|---|
| **Internationalization** | 13 |
| **Multithreaded correctness** | 3 |
| **Malicious code vulnerability** | 13 |
| **Correctness** | 3 |

**Tabela X.** Identificação da quantidade de vezes em que cada bug ocorre na ferramenta SpotBugs.

#### 10.1.2.1 Análise dos Bad Smells:

- **Dodgy code:** trata de código confuso, vulnerável a erros e com comportamento possivelmente anômalo, desta categoria foram encontrados 9 bugs. Elas subdividem-se em 3 subcategorias: “Redundant comparison to null”, “Misuse of static fields” e “Unread field”.
    - **Redundant comparison to null:** ocorreu 2 vezes comparações redundantes de variáveis não nulas à constante Null.
    - **Misuse of static field:** ocorreu 6 vezes escritas a campos estáticos, geralmente isto indica uma má prática.
    - **Unread field:** ocorreu 1 vez a presença de um campo protected que nunca é lido.

- **Performance:** implica no bom uso de alocação de memória e uso de classes não estáticas. Foram identificados 10 bugs divididos nas seguintes 4 subcategorias:
    - **Questionable Boxing of primitive value:** ocorreu 1 vez a invocação de “new Long” ao invés do método mais eficiente “valueOf()”. Além de ocorrer 2 vezes conversões de primitivas para extração do valor real ao invés do uso de métodos parse.
    - **Unread field:** ocorreu 5 vezes a presença de campos nunca lidos.
    - **Inner class could be made static:** ocorreu 1 vez a presença de uma classe interna que poderia ser tornada estática, tornando instâncias da classe externa menores.
    - **Unused field:** ocorreu 1 vez a presença de campos nunca utilizados.

- **Bad practice:** faz jus a más práticas de programação que acarretam diversos problemas de manutenibilidade e evolução de código no longo prazo. Desta categoria foram encontrados 7 bugs, subdividos nas seguintes 5 subcategorias:
    - **Problems with implementation of equals():** está presente 1 bug em que a classe implementa CompareTo, porém herda o “equals” de “Object”, ao invés de reimplementá-lo.
    - **Stream not closed on all paths:** ocorre 1 bug em que o fluxo de entrada não é fechado após o seu uso, podendo acarretar um vazamento de um descritor de arquivo.

    - **Bad use of return value form method:** é indicado 1 bug em que é ignorado o retorno de um método, isto pode levar a um comportamento inadequado.
    - **Dubious method invocation:** está presente 1 bug em que um entrySet só é válido durante determinada iteração de um mapa, seu uso em um addAll pode levar a comportamento não determinístico.
    - **Incorrect definition:** ocorre 1 bug em que um atributo da classe não é serializável, apesar da classe implementar Serializable, isto pode levar a problemas neste atributo ao deserializar a classe.

- **Internacionalization:** trata de formas de internacionalização do código, que incluem uso adequados de codificação. Nesta categoria estão presentes 13 bugs subdivididos em na seguinte subcategoria:
    - **Dubious method used:** são indicados 13 bugs em que um fluxo de entrada é convertido diretamente de byte para String, sem indicação da codificação a ser utilizada. Isto pode causa comportamento não determinístico para diferentes plataformas.
- **Multithreaded correctness:** indica problemas de sincronização entre threads. Nesta categoria ocorrem 3 bugs, todos estão inseridos na subcategoria a seguir:
    - **Inconsistent synchronization:** presentes 3 bugs que indicam que há um possível uso inconsistente de sincronização como o acesso de métodos travados e destravados na mesma classe ou um acesso travado realizado pelo próprio método da classe.
- **Malicious code vulnerability:** indica exposição de classes e atributos a outrem que não deveriam ter acesso a estes. Foram detectados 13 bugs, eles subdividem-se em 3 subcategorias:
    - **Storing reference to mutable object:** ocorre 5 bugs, em que é atribuído à representação interna um objeto mutável externo ao invés de que seja armazenada uma cópia do objeto, essa atribuição se feita por um código inseguro pode comprometer as informações internas do objeto.
    - **Method returning array may expose internal representation:** foram detectados 5 bugs em que é retornado a representação interna de um objeto ao invés de retornar uma cópia do objeto, caso o retorno seja enviado a um código inseguro isso pode comprometer a segurança.
    - **Mutable static field:** nesta subcategoria foram identificados 3 bugs descritos a seguir: o primeiro consiste em um atributo mutável estático, que pode ser alterado por um código malicioso; o segundo se trata de da atribuição de uma coleção mutável a um atributo estático, um código malicioso ou por acidente de uma biblioteca externa podem fazer uso dessa vulnerabilidade; o último bug diz respeito a um campo estático público que não é final, podendo ser alterado por um código externo mal-intencionado.
- **Correctness:** esta categoria trata da corretude do código e apresenta problemas claros de codificação, nela foram encontrados 3 bugs divididos em 3 subcategorias:

- **Infinte Loop:** trata de 1 bug em que foi detectado a presença de um loop infinito, resultando eventualmente em um stackoverflow.
- **Redundant comparison to null:** diz respeito à 1 bug em que a verificação se o valor é nulo é redundante, pois este já foi verificado anteriormente, isto indica que não é possível que este valor seja nulo (a não ser que o código de verificação anterior seja errôneo).
- **Null pointer dereference:** apresenta-se com 1 bug, indicando que um parâmetro nulo foi enviado a um método que deveria receber um parâmetro não nulo.

Dentre todos os bugs descritos acima, é possível definir que os mais preocupantes são dos tipos: “Multithreaded correctness”, por levar a eventuais problemas de incoerência no contexto multithreaded; “Malicious code vulnerability”, indica problemas de segurança presentes principalmente em atributos; “Bad practice”, por indicar problemas no fluxo de entrada do código, definição inconsistente de variável em classes serializáveis e na implementação de “equals” em um objeto, estas falhas tornam o código suscetível a um comportamento indesejado. O projeto na totalidade apresenta várias falhas que precisam ser corrigidas, porém, algumas estão bem intrínsecas

### 10.1.3 Checkstyle:

“Checkstyle” é uma ferramenta que verifica o estilo do código no projeto de acordo com um padrão pré-estabelecido. Como desconhecemos os padrões do “web-magic”, usamos os padrões já embutidos na ferramenta. Dito isso, escolhemos o padrão “Google Checks”, que identificou 7427 irregularidades no projeto, com 35 tipos distintos dentre 176 arquivos. Note que, com irregularidades, nos referimos ao código que funciona, porém, está mal escrito.

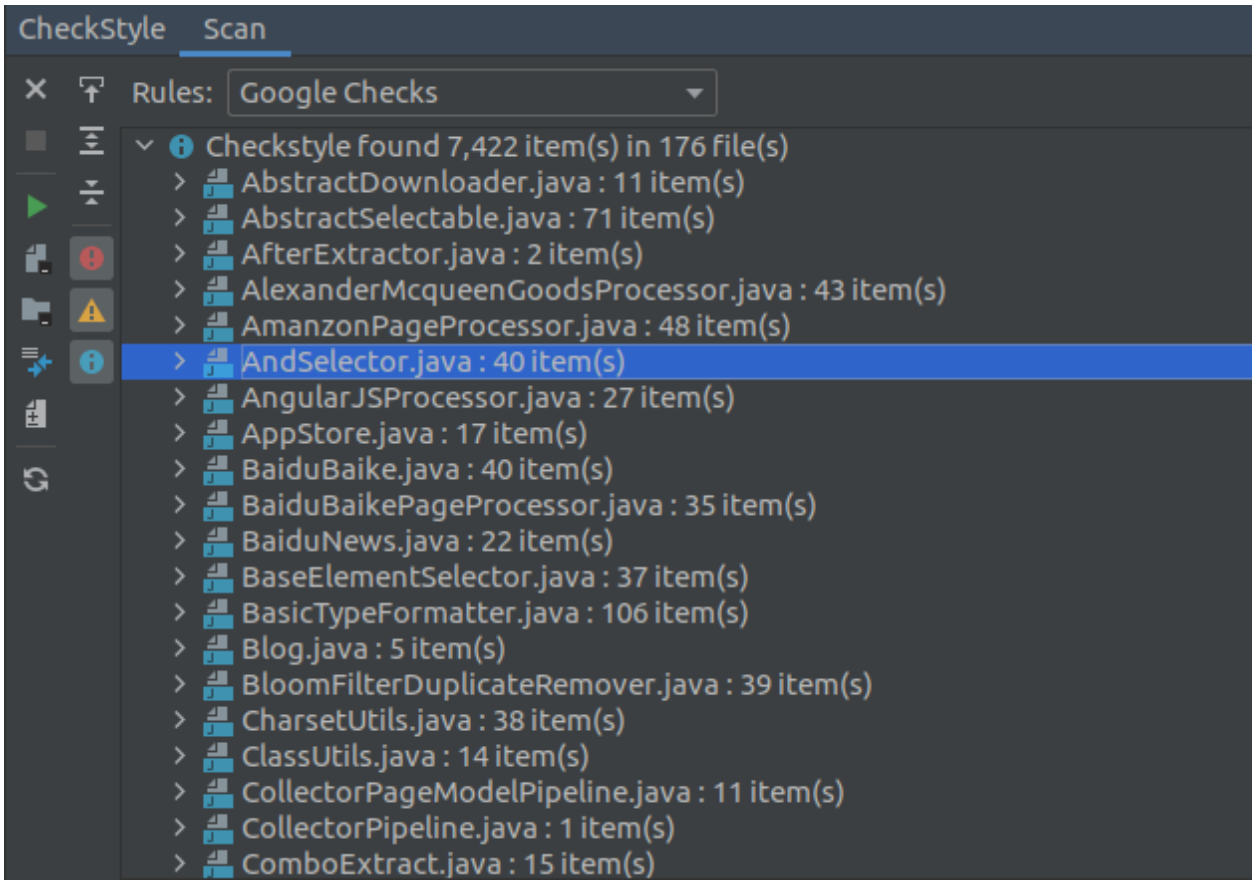


**Figura X.** Irregularidades de códigos encontrados pelo padrão de estilo de código “Google Checks”.

| | |
|---|---|
| **Indentation** | 5793 |
| **CustomImportOrder** | 353 |
| **FileTabCharacter** | 340 |
| **LineLength** | 197 |
| **SummaryJavadoc** | 192 |

**Tabela X**: Identificação dos cinco maiores problemas de estilo no código segundo o padrão “Google Checks”.

- **Indentation:** representa diferença entre a indentação esperada e a real. No caso, era esperada uma indentação de 2 espaços, mas o projeto usou a indentação de 4 espaços.

- **CustomImportOrder:** indica que a ordem dos pacotes importados estão como o esperado. Aqui é indicado que a forma correta de ordenar os imports seria primeiro importar os pacotes nativos do java e só depois os pacotes de terceiros.
- **OverloadMethodsDeclarationOrder:** trata dos métodos sobrecarregados estarem agrupados, nele um erro de estilo é acionado se houver dois métodos de mesmo tipo (nome, parâmetros e retorno) separados por outros métodos.
- **SummaryJavadoc:** representa a presença de um resumo da classe ou método antes dos argumentos.
- **MissingJavadocMethod:** indica que o comentário javadoc não está presente em cada classe e método.
- **WhitespaceAfter:** verifica se todas as separações pontuais são seguidas de espaço em branco.
- **LineLength:** indica se o tamanho da linha deve ser menor ou igual a 100.
- **OperatorWrap:** trata da ocorrência de uma concatenação multilinha, quando o operador de concatenação "+" não está na nova linha.
- **WhitespaceAround:** ocorre quando os delimitadores de métodos ou classes "{}" não são antecedidos por espaço.
- **FileTabCharacter:** diz respeito a indentação feita com tab ('/t') ao invés de espaço.
- **RequireEmptyLineBeforeBlockTagGroup:** indica que no javadoc, não há uma linha separando o resumo do parâmetro que vem seguida.
- **AbbreviationAsWordinName**: trata de que o estilo nomenclatura “CamelCase” não foi respeitado ao nomear uma classe, método ou variável.
- **LeftCurly**: verifica se a abertura de um método “{” não está na mesma linha que a assinatura, independente do tamanho da linha.
- **ParenPad**: indica se há um espaço logo após a abertura de parênteses, por exemplo, um for com espaço à esquerda.
- **NoWhitespaceBefore**: diz respeito à presença de um espaço em branco antes de um separador.

Dados estas falhas, podemos concluir que se o GoogleChecks estivesse configurado com indentação de 4 espaços, o projeto teria bem menos irregularidades. As cinco falhas mais presentes do CheckStyle estão relacionadas a detalhes comumente ignorados que não possuem grande impacto legibilidade do código. Ademais, erros como “EmptyCatchBlock” são mais graves, porém ele, por exemplo, aparece apenas uma única vez em todo o código. Com isto, é possível ser otimista em relação à qualidade de estilo do código.

### 10.1.4 PMD:

PMD é um software que analisa o código-fonte e identifica diversas falhas de programação como variáveis declaradas e não utilizadas, blocos vazios, más práticas, estilo de código, performance, entre outros. Os resultados obtidos nesta ferramenta estão expressos na Figura X e na Tabela X, abaixo.

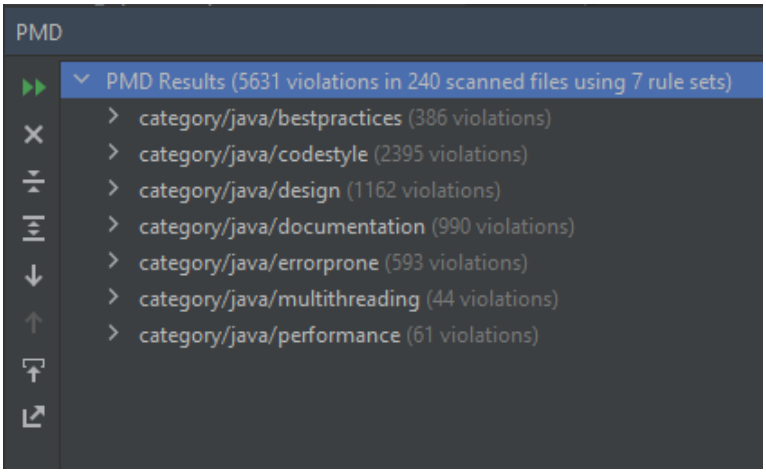


**Figura X:** Categorias e número de violações identificadas no sistema web-magic pela ferramenta.

No total foram identificadas 5631 violações divididas em 7 categorias:

| CATEGORIA | VIOLAÇÕES |
|---|---|
| Best practices | 386 |
| Codestyle | 2395 |
| Design | 1162 |
| Documentation | 990 |

| | |
|---|---|
| Errorprone | 593 |
| Multithreading | 44 |
| Performance | 61 |
| Total | 5631 |

**Tabela X:** Número de violações por categoria no sistema web-magic pela ferramenta.

Entre as violações mais recorrentes e significativas para cada categoria, temos:

**Best practices:**

- **SystemPrintln** - 70 Violações - As referências a System. (Out | err) .print são normalmente destinadas a propósitos de depuração e podem permanecer na base de código, mesmo no código de produção. Ao usar um logger, pode-se ativar / desativar esse comportamento à vontade (e por prioridade) e evitar entupir o log de saída padrão.
- **AccessorMethodGeneration** - 44 Violações - Ao acessar campos / métodos privados de outra classe, o compilador Java irá gerar métodos de acesso com visibilidade privada do pacote. Isso adiciona sobrecarga e para a contagem do método dex no Android. Essa situação pode ser evitada alterando a visibilidade do campo / método de privado para privado de pacote.
- **GuardLogStatement** - 23 Violações - Está uma métrica que verifica se o debug está ativado para logar ou não uma informação.
- **UnusedPrivateField** - 18 Violações - Detecta quando um campo privado é declarado e / ou atribuído a um valor, mas não é usado.
- **AbstractClassWithoutAbstractMethod** - 13 Violações - A classe abstrata não contém nenhum método abstrato. Uma classe abstrata sugere uma implementação incompleta, que deve ser completada por subclasses que implementam os métodos abstratos. Se a classe se destina a ser usada apenas como uma classe base (não para ser instanciada diretamente), um construtor protegido pode ser fornecido para impedir a instanciação direta.
- **LiteralsFirstInComparisons** - 11 Violações - Posicione os literais primeiro em todas as comparações de String, se o segundo argumento for nulo, então NullPointerExceptions podem ser evitados, eles retornarão apenas falso. Observe que alternar as posições literais para compareTo e compareToIgnoreCase pode alterar o resultado, consulte os exemplos.

**Codestyle:**

- **MethodArgumentCouldBeFinal** - 760 Violações - Um argumento de método que nunca é reatribuído dentro do método pode ser declarado final.
- **OnlyOneReturn** - 145 Violações - Um método deve ter apenas um ponto de saída e esse deve ser a última instrução do método.
- **AtLeastOneConstructor** - 126 Violações - Cada classe não estática deve declarar pelo menos um construtor. As classes com membros apenas estáticos são ignoradas.
- **UseDiamondOperator** - 89 Violações - Use o operador diamante para permitir que o tipo seja inferido automaticamente. Com o operador Diamond é possível evitar a duplicação dos parâmetros de tipo. Em vez disso, o compilador agora pode inferir os tipos de parâmetro para chamadas do construtor, o que torna o código também mais legível.
- **LongVariable** - 83 Violações - Campos, argumentos formais ou nomes de variáveis locais que são muito longos podem tornar o código difícil de seguir.
- **MethodNamingConventions** - 26 Violações - Convenções de nomenclatura configuráveis para declarações de método. Esta regra relata declarações de método que não correspondem à regex que se aplica ao seu tipo específico (por exemplo, teste JUnit ou método nativo). Cada regex pode ser configurado por meio de propriedades.
- **LocalVariableNamingConventions** - 3 Violações - Convenções de nomenclatura configuráveis para declarações de variáveis locais e outras variáveis com escopo local. Esta regra relata declarações de variáveis que não correspondem à regex que se aplica ao seu tipo específico (por exemplo, variável final ou parâmetro catch-clause). Cada regex pode ser configurado por meio de propriedades.

**Design:**

- **LawOfDemeter** - 929 Violações - A Lei de Deméter é uma regra simples, que diz "só fale com os amigos". Ajuda a reduzir o acoplamento entre classes ou objetos.
- **ImmutableField** - 115 Violações - Identifica campos privados cujos valores nunca mudam depois que a inicialização do objeto termina na declaração do campo ou por um construtor. Isso ajuda a converter classes existentes em classes imutáveis.
- **CyclomaticComplexity** - 12 Violações - A complexidade dos métodos afeta diretamente os custos de manutenção e a legibilidade. Concentrar muita lógica de decisão em um único método torna seu comportamento difícil de ler e mudar.
- **ClassWithOnlyPrivateConstructorsShouldBeFinal** - 2 Violações - Uma classe com apenas construtores privados deve ser final, a menos que o construtor privado seja invocado por uma classe interna.

**Documentation:**

- **CommentRequired** - 883 Violações - Indica se os comentários javadoc (formais) são necessários (ou indesejados) para elementos específicos da linguagem.
- **CommentSize** - 82 Violações - Determina se as dimensões dos comentários que não são do cabeçalho encontrados estão dentro dos limites especificados.
- **UncommentedEmptyMethodBody** - 18 Violações - Corpo de método vazio não comentado localiza instâncias em que o corpo de um método não contém instruções, mas não há comentários. Ao comentar explicitamente os corpos de métodos vazios, é mais fácil distinguir entre métodos vazios intencionais (comentados) e não intencionais.
- **UncommentedEmptyConstructor** - 7 Violações - Construtor Vazio Não Comentado encontra instâncias onde um construtor não contém instruções, mas não há comentários. Ao comentar explicitamente os construtores vazios, é mais fácil distinguir entre construtores vazios intencionais (comentados) e não intencionais.

**Errorprone:**

- **BeanMembersShouldSerialize** - 269 Violações - Se uma classe é um bean ou é referenciada por um bean direta ou indiretamente, ela precisa ser serializável. As variáveis de membro precisam ser marcadas como transitórias, estáticas ou ter métodos de acesso na classe.
- **AvoidDuplicateLiterals** - 134 Violações - O código que contém literais de String duplicados geralmente pode ser melhorado declarando a String como um campo constante.
- **ReturnEmptyArrayRatherThanNull** - 1 Violações - Para qualquer método que retorne uma matriz, é melhor retornar uma matriz vazia em vez de uma referência nula. Isso remove a necessidade de verificação de nulos em todos os resultados e evita NullPointerExceptions inadvertidas.

**Multithreading:**

- **DoNotUseThreads** - 15 Violações - A especificação J2EE proíbe explicitamente o uso de threads. Threads são recursos que devem ser gerenciados e monitorados pelo servidor J2EE.
- **AvoidSynchronizedAtMethodLevel** - 15 Violações - A sincronização no nível do método pode causar problemas quando um novo código é adicionado ao

método. A sincronização em nível de bloco ajuda a garantir que apenas o código que precisa de sincronização a obtenha.

- **UseConcurrentHashMap** - 12 Violações - Como o Java5 trouxe uma nova implementação do Mapa projetado para acesso multiencadeado, você pode realizar leituras de mapa eficientes sem bloquear outros encadeamentos.

**Performance:**

- **AvoidInstantiatingObjectsInLoops** - 32 Violações - Novos objetos criados dentro de loops devem ser verificados para ver se eles podem ser criados fora deles e reutilizados.
- **RedundantFieldInitializer** - 9 Violações - Java inicializará campos com valores padrão conhecidos, portanto, qualquer inicialização explícita desses mesmos padrões é redundante e resulta em um arquivo de classe maior (aproximadamente três instruções de bytecode adicionais por campo).
- **AvoidFileStream** - 11 Violações - As classes FileInputStream e FileOutputStream contêm um método finalizador que causará pausas na coleta de lixo.

Essas violações possuem níveis de prioridade pré-definidas pelo PMD, na tabela abaixo temos a quantidade de violações para cada categoria de prioridade.

| Prioridade | Quantidade de violações |
|---|---|
| Alta | Codestyle: 87<br>Design: 15<br>ErrorProne: 2<br>Performance: 11 |
| Média | Best Pratices: 10<br>Codestyle: 6<br>ErrorProne: 134 |
| Baixa | Best Pratices: 376<br>Codestyle: 2302<br>Design: 1147<br>Documentation: 990<br>Errorprone: 457<br>Multhreading: 44<br>Performance: 51 |

**Tabela X:** Agrupamento de violações por gravidade do erro (em Alta, Média ou Baixa).

A seguir serão tratados os principais problemas identificados nos 3 níveis de gravidade produzidos pela ferramenta na tabela acima (gravidade alta, média e baixa):

- **Alta:** temos que estas violações costumam ser prejudiciais ao projeto por provocar efeitos anômalos e por causar erros mais graves.

  Dentre este tipo erro em destaque temos o “ReturnEmptyArrayRatherThanNull” da categoria “Errorprone”. Este erro ocorre quando um método que retorna o tipo array, retornar null invés de um array vazio, isso aumenta a chance de causar erro no software se não houver um tratamento adequado para a situação de null em todas as chamadas do método.

  De modo similar na categoria de “Perfomance”, foi indicada a violação “AvoidFileStream”, ela identificou o uso inadequado das classes “FileInputStream” e “FileOutputStream”. Ambas causam pausas no “Garbage Collector” de Java, por esta razão, podem se tornar um grande gargalo no sistema se utilizadas com frequência. Para contornar esta situação elas não devem utilizar seu próprio construtor, devem ser utilizadas através dos métodos “Files.newInputStream()” e “Files.newOutputStream()”, respectivamente.
- **Média:** encontram-se problemas que podem causar comportamento anômalo no código por más práticas no código.

  Dentre elas, em sua maioria estão no campo de “Errorprone” e se tratam de “DataFlowAnomalyAnalysis”, ou seja, são problemas como atribuir duas vezes o valor a mesma variável, chamadas variáveis indefinidas.

  Em CodeStyle, foram encontradas 6 violações com prioridade média, tratam-se de “ShortClassName”, elas apontam que existem nomes de classes com menos de 6 caracteres. Alguns exemplos destas classes são: “Page”, “Site”, “Json”, “Blog” e “Task”. Esta métrica, apesar da prioridade média, não causa falhas na execução e pode ser tratada de modo opcional, conforme as particularidades do projeto.
- **Baixa:** está presente em maior volume, estas violações tratam-se de problemas de formatação e fatores que influenciam na legibilidade do código, geralmente não causam problemas graves e alguns são irrelevantes ou subjetivos.

  Estão concentrados principalmente na categoria de "Codestyle”. Nela temos violações como “LocalVariableCouldBefinal”, esta métrica alerta que variáveis locais que são utilizadas somente uma vez poderiam ser declaradas com Final. Entretanto, não é porque uma variável foi usada apenas uma vez, que necessariamente queremos marcar ela como “final”, em algumas situações pode ser bom para garantir a imutabilidade de um dado específico. Sendo assim, este uso obrigatório do “final” no código em geral é discutível e o mesmo vale para violações de prioridade baixa com estruturas similares, como “MethodArgumentCouldBeFinal”.

### 10.1.5 Considerações Finais:

Nesta seção serão tratadas resumos sobre quais resultados convergiram, divergiram e qual a nossa conclusão sobre o sistema de software, baseando-se nas métricas obtidas da análise estática nas seções 10.1.2, 10.1.3 e 10.1.4.

#### 10.1.5.1 Convergência entre Ferramentas:

**Sobre estilo de código** as seguintes violações convergiram:

- O “PMD” e o “Checkstyle” convergiram, no que se trata da quantidade de condicionais if/else sem chaves, ambas ferramentas apontam 25 ocorrências em suas respectivas métricas, chamadas “NeedBraces” e “IfStmtsMustUseBraces”.
- As métricas “MethodNamingConventions” e “AvoidDollarSigns” do “PMD” apontam a uma falha semelhante a indicada “MethodName” do “CheckStyle”. Elas indicam que o nome do método deve seguir o padrão ‘^[a-z][a-z0-9][a-zA-Z0-9_]*$'. No caso, o método possui um “$” no nome.
- A violação “PrematureDeclaration” do “PMD” equivale à “ViariableDeclarationUsageDistance” do “CheckStyke”, as ferramentas encontraram uma e duas ocorrências, respectivamente.

**Sobre más práticas,** as violações abaixo convergiram:

- A métrica “Inconsistent synchronization” do “Bad Smells” aponta a mesma falha que a métrica “AvoidSynchronizedAtMethodLevel” do “PMD”, elas indicam que o uso de multithreading é inconsistente, este tipo de falha pode levar a comportamento anômalo em tempo de execução.
- As violações “UnusedField” e “UnreadFields” da ferramenta “Bad Smells” apontam uma falha similar à falha do PMD no campo “UnusedPrivateField“. Estas falhas indicam haver variáveis definidas com um valor atribuído, porém estas variáveis nunca são lidas. Na primeira ferramenta isto é apontado como uma violação de performance e de código “duvidoso”, a última ferramenta aponta como uma falha de más práticas.

#### 10.1.5.2 Divergência entre Ferramentas:

**Sobre estilo de código** as seguintes violações divergiram:

- O “PMD” possui métricas de estilo que checam irregularidades no tamanho dos nomes declarados para métodos e classes, acusando problema quando, por exemplo, estes nomes são muito curtos. Por outro lado, a ferramenta “Checkstyle” não identifica este tipo de violação como parte de suas checagens padrões.
- A métrica “MissingSwitchDefault” do “Checkstyle” ocorreu uma vez no projeto, apontando que existe um condicional switch que não possui um valor default. Entretanto, o “PMD” não apontou nenhuma nenhuma ocorrência de sua métrica equivalente presente na coleção de regras “best pratices” chamada “DefaultLabelNotLastInSwitch”.
- A situação descrita acima se repete em outras checagens, como no caso da métrica “EmptyCatchBlock” que é acusada uma vez pelo “Checkstyle” e nenhuma pelo “PMD**”**, que realiza a mesma verificação.

**Sobre más práticas:**

- Dentre as violações mais graves identificáveis em ambas as ferramentas "PMD" e “Bad Smells”, a violação “ReturnEmptyArrayRatherThanNull” foi relatada na primeira, porém não na segunda. Por poder causar um erro grave em tempo de execução (ao retornar nulo ao invés de um array vazio), a presença dessa violação indica um erro grave.
- A violação “AvoidFileStream” foi indicada exclusivamente na ferramenta “PMD” e nenhuma semelhante foi identificada na ferramenta “Bad Smells”, ela trata de um problema grave de performance encontrada no uso dos construtores das classes “InputFileStream” e “OutputFileStream”, causando pausas constantes no “Garbage Collector” de "Java".

#### 10.1.5.3 Conclusão sobre a análise estática:

A partir da análise das métricas descritas e registradas acima é possível assumir que: o projeto é de baixo a intermediariamente complexo pelo número de classes, de linhas e pela sua complexidade ciclomática; possui um número exagerado de *override*; o padrão especialista da informação é afetado por possuir atributos visíveis às outras classes graças ao constante de herança; o fluxo de execução é conciso; a documentação javadoc é satisfatória nas classes e métodos; o projeto apresenta um valor aceitável de testes, porém um pouco abaixo do esperado. As ferramentas expuseram diversas situações sensíveis do projeto, erros graves e passíveis de quebra.

## 11. Testes

## 12. Evolução

## 13. Conclusões