

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Toni Marcos Schmitt

Refatoração da Ferramenta CodeMaster

Florianópolis - SC

2022

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Toni Marcos Schmitt

Refatoração da Ferramenta CodeMaster

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para a obtenção do Grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Jean Carlo Rossa Hauck

Florianópolis - SC

2022

Toni Marcos Schmitt

Refatoração da Ferramenta CodeMaster

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Dr. Jean Carlo Rossa Hauck

Orientador

Universidade Federal de Santa Catarina

Prof.^a Dr.^a rer. nat. Christiane Gresse von Wangenheim, PMP

Universidade Federal de Santa Catarina

Prof. Dr. Frank Augusto Siqueira

Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus pais,
Adenir e Maria, a minha companheira Barbara
e nossa amada filha Sara.

Agradeço ao meu orientador, profº Jean, por ter me aceito como orientando, por toda paciência a mim dedicada e por sempre ter me incentivado a continuar o trabalho.

Agradecimento especial aos meus pais, Adenir e Maria, por todo apoio recebido, à minha companheira Barbara, por todo incentivo, amor e companheirismo, a nossa amada filha Sara, e a todos que me apoiaram direta ou indiretamente para conclusão deste trabalho.

RESUMO

Com a introdução e avanço de softwares em todas as áreas da sociedade, a demanda por profissionais capacitados para seu desenvolvimento também cresce. Surge então a necessidade de inserir o ensino de computação na educação básica, para que as jovens se sintam mais próximas e interessadas no desenvolvimento de softwares. Para facilitar a inserção de usuários leigos ao universo do desenvolvimento de softwares, existem ferramentas de programação baseadas em blocos, como o App Inventor, que possibilitam o desenvolvimento de apps para dispositivos móveis para Android. Neste contexto, a ferramenta CodeMaster visa apoiar na avaliação de projetos desenvolvidos pelos alunos. A versão original do CodeMaster analisa e avalia projetos desenvolvidos com o App Inventor e Snap!, além de imagens. Deste modo, o objetivo deste trabalho é realizar a refatoração de código do CodeMaster, atualizando algumas tecnologias defasadas que são utilizadas em seu código fonte, como *Servlets* e *JSP* e, separando as camadas de *back-end* e *front-end*. Com esta manutenção perfectiva, utilizando *frameworks* atuais para o sistema, espera-se facilitar a manutenibilidade do código e o desenvolvimento de novas funcionalidades no futuro.

Palavras-chaves: *CodeMaster*, Manutenção perfectiva, Refatoração de código.

LISTA DE SIGLAS

JSP - *Java Server Pages*

HTML - *HyperText Markup Language*

CSS - *Cascading Style Sheets*

JPA - *Java Persistence API*

XP - *eXtreme Programming*

API - *Application Programming Interface*

REST - *Representational State Transfer*

Web - *World Wide Web*

GQS - *Grupo de Qualidade de Software*

LISTA DE FIGURAS

Figura 1 - Diagrama de Casos de Uso CodeMaster.....	19
Figura 2 - Página inicial do CodeMaster (DEMETRIO, 2017).....	21
Figura 3 - Processo geral de reengenharia de software (SOMMERVILLE, 2011).....	24
Figura 4 - Diagrama de componentes (PELLE, 2018).....	31
Figura 5 - Código do resultado de avaliação do código Snap!.....	36
Figura 6 - Tela de resultado da avaliação de projeto Snap!.....	37
Figura 7 - Tela de resultado da avaliação de Interface Gráfica de projeto App Inventor.....	38
Figura 8 - Cadastro e Estatísticas para Pesquisadores.....	39
Figura 9 - Consulta para Pesquisadores.....	39
Figura 10 - Área de estética na interface de usuário.....	40
Figura 11 - Código do menu do CodeMaster legado.....	45
Figura 12 - Novo HTML do menu do CodeMaster V2.0.....	46
Figura 13 - Novo TypeScript do menu do CodeMaster V2.0.....	47
Figura 14 - Consulta de turmas de professor no CodeMaster legado.....	48
Figura 15 - Nova consulta de turmas de professor no CodeMaster.....	49
Figura 16 - Nova consulta por identificador.....	49
Figura 17 - Insert de Turma no CodeMaster legado.....	50
Figura 18 - Criação da instância da Turma no CodeMaster.....	50
Figura 19 - Insert da instância da Turma na base de dados do CodeMaster.....	50
Figura 20 - Upload de projeto App Inventor no CodeMaster.....	55
Figura 21 - Resultado de programação de projeto App Inventor no CodeMaster.....	56
Figura 22 - Resultado de Interface Gráfica de projeto App Inventor no CodeMaster.....	58
Figura 23 - Upload de projeto Snap! no CodeMaster.....	58
Figura 24 - Resultado de projeto Snap! no CodeMaster.....	59
Figura 25 - Criação de turmas App Inventor do CodeMaster.....	60
Figura 26 - Seleção de conceitos para avaliação do App Inventor do CodeMaster.....	61
Figura 27 - Upload dos arquivos App Inventor da turma do CodeMaster.....	62
Figura 28 - Resultado da turma App Inventor do CodeMaster.....	63
Figura 29 - Criação de turmas Snap! do CodeMaster.....	64
Figura 30 - Seleção de conceitos para avaliação do Snap! do CodeMaster.....	65
Figura 31 - Upload dos arquivos Snap! da turma do CodeMaster.....	66
Figura 32 - Resultado dos projetos Snap! da turma do CodeMaster.....	67
Figura 33 - Tela inicial do menu Professor do CodeMaster.....	68
Figura 34 - Tela de cadastro de Professor do CodeMaster.....	69
Figura 35 - Tela de login de Professor do CodeMaster.....	70
Figura 36 - Tela do Professor logado no CodeMaster.....	71

Figura 37 - Tela após o login do Administrador no CodeMaster.....	72
Figura 38 - Tela do Administrador logado	73
Figura 39 - Tela após o logout do Professor no CodeMaster.	74
Figura 40 - Tela do Professor logado no CodeMaster.....	75
Figura 41 - Apresentação dos resultados das turmas no CodeMaster.	76

LISTA DE QUADROS

Quadro 1 - Tecnologias utilizadas no CodeMaster (DEMETRIO, 2017).....	32
Quadro 2 - Requisitos Funcionais do CodeMaster e suas evoluções.....	54

Sumário

1 INTRODUÇÃO.....	13
1.2 OBJETIVOS	15
1.2.1 OBJETIVO GERAL	15
1.2.2 OBJETIVOS ESPECÍFICOS	15
1.2.3 MÉTODO DE PESQUISA	15
1.2.3 ESTRUTURA DESTE DOCUMENTO	16
2 FUNDAMENTAÇÃO TEÓRICA.....	17
2.1 Avaliação de Código no Ensino de Computação	17
2.2 CODEMASTER	18
2.2.2 Avaliação no CodeMaster	18
2.2.3 Tecnologias utilizadas:.....	21
2.3 Manutenção de Software.....	22
2.3.1 Manutenção Perfectiva	23
2.4 Refatoração.....	26
3 REFATORAÇÃO DO CODEMASTER	30
3.1 ANÁLISE DA ARQUITETURA ATUAL	30
4.1 ANÁLISE DAS PRINCIPAIS NECESSIDADES DE MANUTENÇÃO	33
4.1.1 Modernização das tecnologias:.....	33
4.1.2 Separação de camadas	33
4.1.3 Inclusão de diversas novas funcionalidades ao longo do tempo	36
4.2 Seleção de novas tecnologias	41
4.2.1 Critérios de seleção	41
4.2.2 Seleção das novas tecnologias.....	41
4.3 IMPLEMENTAÇÃO DA MANUTENÇÃO	42
4.3.1 Configuração do ambiente de desenvolvimento.....	42
4.3.2 Principais dificuldades da manutenção de código	42
4.3.2 Detalhamento das modificações	44
5 TESTES FUNCIONAIS	52

5.1 Requisitos Funcionais do Codemaster	52
5.2 Testes dos Requisitos Funcionais	54
5.2.1 Teste de funcionalidade do menu Aluno - App Inventor	54
5.2.2 Teste de funcionalidade do menu Aluno - Snap!.....	58
5.2.3 Teste de funcionalidade do menu Professor - App Inventor	59
5.2.4 Teste de funcionalidade do menu Professor - Snap!.....	63
5.2.5 Teste da funcionalidade cadastrar Professor	67
5.2.6 Teste da funcionalidade login Professor	69
5.2.7 Teste da funcionalidade login de Administrador	71
5.2.9 Teste da funcionalidade logout Professor	73
5.2.10 Teste da funcionalidade logout de Administrador	74
5.2.11 Teste da funcionalidade manter registros da avaliação anônima	74
5.2.12 Teste da funcionalidade manter registros das avaliação do Professor	75
5.2.13 Teste da funcionalidade apresentar estatísticas de avaliações	76
6 CONCLUSÃO	77
REFERÊNCIAS	79
APÊNDICE A – Código-Fonte	84
APÊNDICE B - Artigo	85

1 INTRODUÇÃO

Com a evolução das tecnologias computacionais e dos dispositivos embarcados, para se tornarem consumidores mais produtivos e até criadores de sistemas computacionais, as pessoas necessitam do conhecimento básico sobre os princípios da computação (CSTA, 2017). Por este motivo, torna-se cada vez mais importante o ensino de computação nas escolas.

Entretanto, a inclusão do ensino de computação nas escolas esbarra, dentre outros fatores, na falta de ferramentas de suporte aos professores na avaliação de trabalhos práticos de programação dos estudantes (MORENO-LÉON et al., 2015). Esta situação é agravada pela falta de formação dos professores do Ensino Básico em áreas relacionadas a tecnologia da informação e comunicação. Esta lacuna gera um grande esforço para colocar em prática o ensino de computação na escola, seja na definição das atividades e até mesmo na avaliação das atividades práticas desenvolvidas pelos alunos (ESERYEL et al., 2013).

Porém este cenário está se modificando. Atualmente há iniciativas que entendem que todos os alunos em todas as escolas devem ter a oportunidade de aprender computação e visam aumentar o ensino de computação no Ensino Fundamental e Médio (CNE, 2019). Uma das formas de ensinar programação é por meio de trabalhos práticos, em que os alunos desenvolvem programas utilizando linguagens de programação. Neste contexto, pode-se utilizar as ferramentas App Inventor (APP INVENTOR, 2022) e Snap! (SNAP, 2022) entre outros, que são ambientes para programação visual baseada em blocos que permite que pessoas, sem conhecimento computacional, comecem a programar e construir aplicativos para dispositivos Android ou ferramentas similares que possuem o mesmo objetivo.

No contexto de falta de formação específica dos professores, a ferramenta CodeMaster (DEMETRIO, 2017) auxilia na avaliação dos trabalhos desenvolvidos pelos

alunos, permitindo a avaliação automatizada de vários conceitos aumentando a objetividade e eliminando qualquer favoritismo ou inconsistência (ZEN et al., 2011).

Entretanto, a ferramenta CodeMaster foi inicialmente criada em 2016 e, desde então, diversas funcionalidades adicionais foram sendo implementadas por outros trabalhos de pesquisa, tais como (Pelle, 2018; Justen, 2019; Martins, 2019). Assim, com a incorporação de diversas funcionalidades, o código-fonte sofreu degradação, perdendo sua manutenibilidade. Manutenibilidade é a facilidade com que um sistema ou componente de *software* pode ser modificado para alterar ou adicionar recursos, corrigir falhas ou defeitos, melhorar o desempenho ou outros atributos ou adaptar-se a um ambiente alterado (ISO/IEC/IEEE 24765, 2017).

Além disso, algumas das tecnologias originalmente utilizadas no desenvolvimento do CodeMaster, como *Servlet*, *JSP* e *Java Script* puro (DEMETRIO, 2017), após mais de seis anos, já se encontram defasadas e são de difícil manutenção. Há também, algumas necessidades de melhoria na arquitetura da ferramenta CodeMaster, especialmente na separação entre *back-end* e *front-end*.

Assim, surge a necessidade de realizar um processo de manutenção perfectiva, que consiste na manutenção de *software* realizada para melhorar o desempenho, manutenibilidade ou outros atributos de um programa de computador (ISO/IEC/IEEE 24765, 2017). As principais manutenções perfectivas percebidas como necessárias são: modernização das tecnologias utilizadas e realização de uma refatoração separando as camadas de *back-end* e *front-end*. Como se trata de uma refatoração, não são realizadas alterações funcionais, portanto os requisitos funcionais contemplados pelo CodeMaster são mantidos (DEMETRIO, 2017).

Espera-se que com a refatoração do sistema CodeMaster, seja melhorada a sua manutenibilidade e a incorporação de novas tecnologias possa facilitar o desenvolvimento de novas funcionalidades no futuro.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O presente trabalho tem como objetivo principal realizar a manutenção perfectiva da ferramenta web de análise e avaliação automatizada CodeMaster, melhorando a manutenibilidade e atualizando as tecnologias utilizadas no código-fonte.

1.2.2 OBJETIVOS ESPECÍFICOS

- Sintetizar a fundamentação teórica sobre a ferramenta CodeMaster, refatoração e manutenção perfectiva.
- Refatorar a ferramenta CodeMaster, incluindo a seleção e implementação de novas tecnologias.
- Testar os requisitos atuais da aplicação em ambiente de desenvolvimento.

1.2.3 MÉTODO DE PESQUISA

A metodologia de pesquisa utilizada para o desenvolvimento deste trabalho é composta pelas seguintes etapas:

Etapa 1 - Fundamentação Teórica.

Nesta etapa, será realizada a análise da literatura referente aos principais conceitos que serão abordados neste trabalho, a etapa será dividida nas seguintes atividades:

A1.1 - Análise teórica sobre o CodeMaster.

A1.2 - Análise teórica sobre manutenção de *software*.

A1.3 - Análise teórica sobre refatoração.

Etapa 3 - Seleção de Tecnologias.

Nesta etapa será realizada a pesquisa e seleção de tecnologias atuais para desenvolvimento Web com Java, levando em conta os requisitos funcionais e não-funcionais da atual versão do CodeMaster.

A3.1 - Análise dos requisitos atuais do CodeMaster.

A3.2 - Análise das restrições de infraestrutura do ambiente de produção do CodeMaster.

A3.3 - Análise do código-fonte atual.

A3.4 - Implementação das melhorias.

Etapa 4 - Testes.

Nesta etapa, são realizados os testes dos requisitos funcionais da ferramenta CodeMaster:

A4.1 - Testar os requisitos funcionais.

A4.2 - Documentar os resultados.

1.2.3 ESTRUTURA DESTE DOCUMENTO

Na seção 2 deste trabalho são abordados os conceitos básicos do CodeMaster, os aspectos tipicamente analisados na avaliação de código no ensino de computação, funcionalidades do *software* e tecnologias utilizadas. Também são apresentados alguns conceitos básicos de manutenção de *software*.

Na seção 4 é apresentada a análise da arquitetura atual do *software*, a análise das principais necessidades de manutenção, requisitos e seleção das novas tecnologias a serem utilizadas, implementação da manutenção perfectiva, principais dificuldades encontradas, detalhamento e comparativo das modificações realizadas no *software*.

Na seção 5 serão apresentados os testes realizados após a refatoração do sistema.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos teóricos referentes a este trabalho. Primeiramente será apresentado o CodeMaster (DEMETRIO, 2017), os conceitos de avaliação de código no ensino de computação, a avaliação no CodeMaster e as tecnologias utilizadas na ferramenta. Apresenta conceitos de manutenção de *software*, desde a manutenção para reparar falhas, adicionar ou modificar funcionalidades além da manutenção perfectiva, para melhorar a manutenibilidade ou desempenho do *software*.

2.1 Avaliação de Código no Ensino de Computação

A análise de código aplicada no CodeMaster é focada no grau de atendimento de objetivos de aprendizagem. Entretanto, não há uma única definição de quais aspectos devem ser analisados, então esses aspectos devem ser derivados diretamente dos objetivos de aprendizagem. A análise e avaliação de código no CodeMaster exige flexibilidade para que o analisador possa fazer uma análise livre do código, pois podem haver diversas soluções corretas para um mesmo problema (DEMETRIO, 2017).

Neste contexto, necessita-se de uma análise e avaliação do programa baseado em atributos mensuráveis que podem ser extraídos do programa, avaliando assim o aprendizado esperado dos alunos. Esta métrica, chamada de rubrica, utiliza medidas descritivas para separar os níveis de desempenho em uma determinada tarefa, delineando os vários critérios associados às atividades de aprendizado (WHITTAKER et al., 2001), e permite o feedback instrucional, definindo a pontuação de cada critério.

O feedback é importante no contexto educacional para facilitar a aprendizagem (MORENO, 2004), também é uma importante técnica para orientar os alunos a pensarem sobre as respostas e fornecer informações para direcionar a mudança de sua forma de pensar e agir (BILAL et al., 2012).

Feedback pode ser considerado como uma resposta para as ações do aluno, indicando a exatidão das respostas ou agindo de maneira ativa, provendo dicas,

sugestões e exemplos relacionados ao conteúdo (BLACK & WILIAN, 1998). Portanto é de grande importância que ao final da avaliação sejam dados nota e feedback para o aluno.

2.2 CODEMASTER

CodeMaster é uma ferramenta web para análise e avaliação de código baseado em linguagens de programação em blocos para dar suporte ao ensino de computação, inicialmente voltada para aplicativos desenvolvidos com App Inventor (DEMETRIO, 2017) e, posteriormente foi aprimorado para também analisar e avaliar aplicativos desenvolvidos com Snap! (PELLE, 2018).

2.2.2 Avaliação no CodeMaster

A avaliação na ferramenta é baseada na rubrica para avaliar apps proposta por Sherman (2015) no Dr. Scratch (DR. SCRATCH, 2017), e no *framework* do pensamento computacional (BRENNAN & RESNICK, 2012).

Focado no ensino de programação infantil, o CodeMaster visa a ampliação do suporte à avaliação de linguagens de programação baseadas em blocos, semi-automatizando a avaliação de aplicativos desenvolvidos pelo App Inventor (DEMETRIO, 2017) e Snap! (PELLE, 2018).

Atualmente, a ferramenta está em constante desenvolvimento pela iniciativa Computação na Escola/INCoD/INE/UFSC, tendo diversos trabalhos acadêmicos voltados para evolução da ferramenta. Até o presente momento, o CodeMaster possui as seguintes funcionalidades como apresentado na Figura 1.

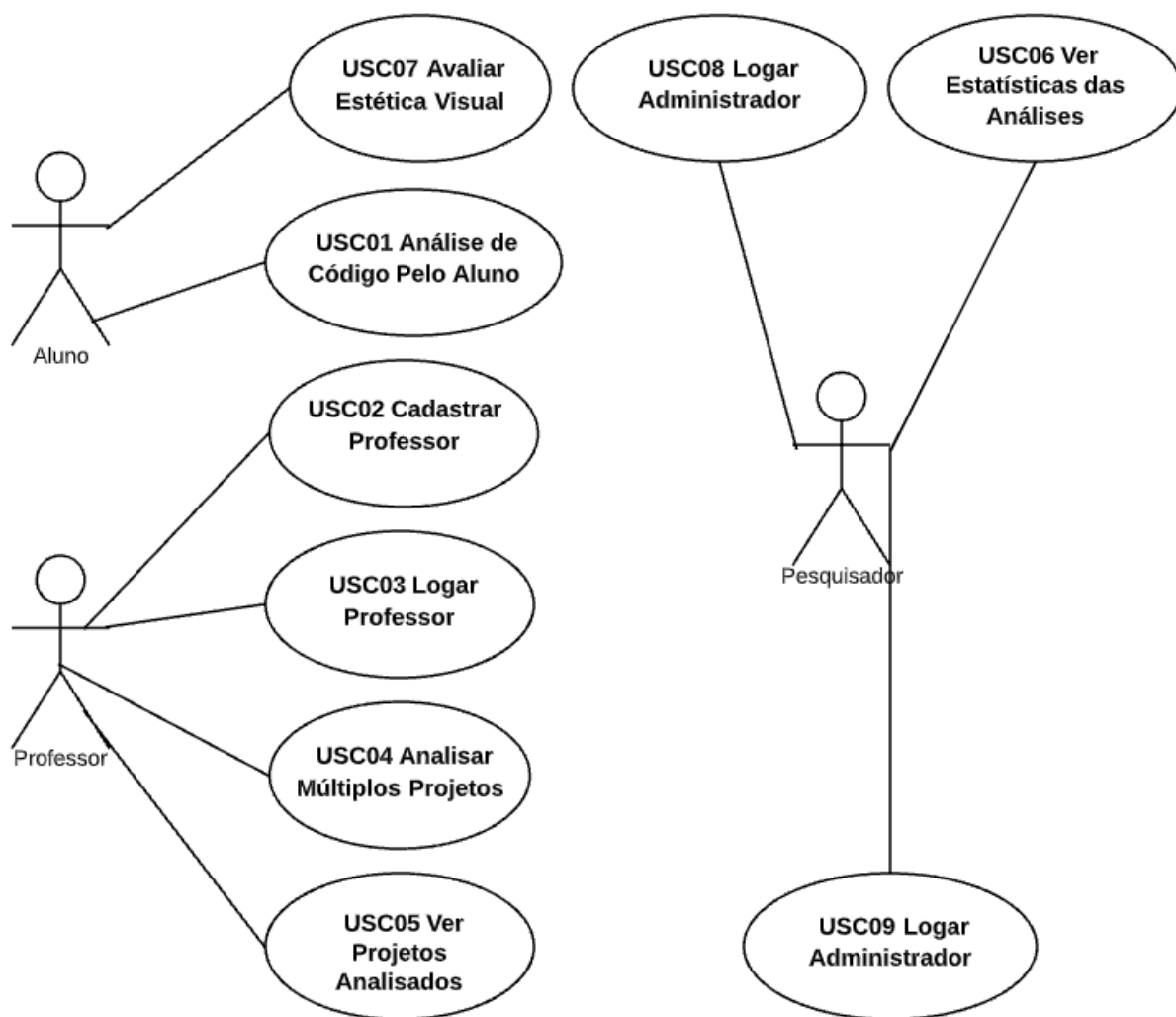


Figura 1 - Diagrama de Casos de Uso CodeMaster

1. *Upload* de código desenvolvido pelo App Inventor ou Snap!, sem necessidade de cadastro prévio, para análise e avaliação automática da ferramenta, apresentando o *feedback* completo da avaliação, incluindo a pontuação por critério analisado, nota obtida e em qual categoria ninja se enquadra.
2. Cadastro do Professor.
 - 2.1. Login do Professor.

- 2.2. Criação de turma por um professor.
- 2.3. Upload de múltiplos projetos desenvolvidos pelo App Inventor ou Snap! para avaliação em lote, apresentando os resultados de forma resumida.
- 2.4. Visualizar turmas e avaliações já realizadas pelo professor, podendo ser exportados.
- 3. Cadastro Administrador
 - 3.1. Login Administrador
 - 3.2. Apresentar todos os dados anônimos sobre as avaliações já realizadas.

O CodeMaster dá suporte a professores e alunos no ensino e aprendizagem de computação no Educação Básica, fornecendo indicadores de corretude, utilidade, funcionalidade e desempenho do código desenvolvido na linguagem de blocos (DEMETRIO, 2017). A Figura 2 apresenta a atual tela inicial do CodeMaster.

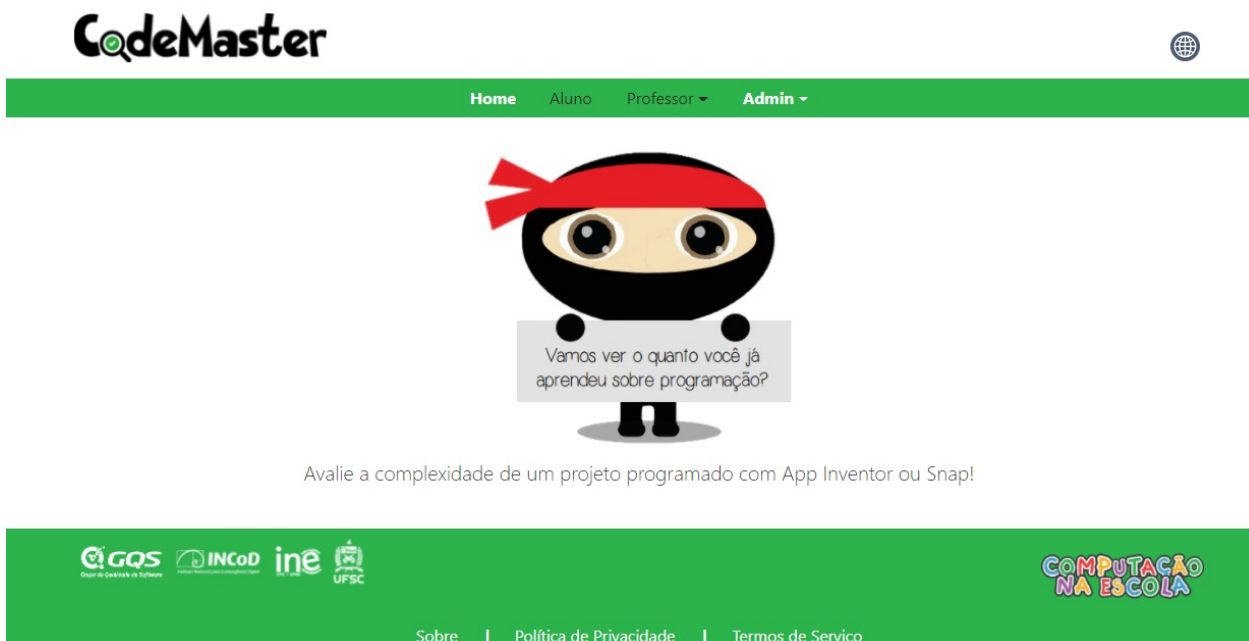


Figura 2 - Página inicial do CodeMaster (DEMETRIO, 2017)

2.2.3 Tecnologias utilizadas:

A ferramenta CodeMaster teve o seu desenvolvimento iniciado no ano de 2016, tendo sido escolhidas tecnologias de domínio dos desenvolvedores iniciais:

- **Java:** É uma linguagem de programação orientada a objetos desenvolvida pela SUN Microsystems na década de noventa.
- **Servlets:** Segundo DEITEL (2004), Servlets estendem a funcionalidade de um servidor, servindo páginas da Web para o navegador do usuário com o protocolo Http, utilizando o conceito de requisição e resposta.
- **JSP:** *Java Server Pages* é uma extensão dos *Servlets*, simplificando o processo de criação de páginas *Web*, separando a apresentação do conteúdo, o *JSP* combina o uso de *HTML* com *Java* e são utilizadas quando parte do conteúdo é estática e parte é gerada dinamicamente pelo código *Java* (DEITEL 2004).

- **Jersey:** *Framework* de código aberto para desenvolvimento de Web Services em *Java* utilizando API JAX-RS, abstraindo detalhes de baixo nível de um serviço REST (JERSEY, 2017).
- **Java Script:** É uma linguagem de programação leve, interpretada e orientada a objetos, roda no *client side* da web, é utilizada para programar o comportamento de uma página web a partir da ocorrência de um evento (MOZILLA, 2022).
- **HTML5:** *Hyper Text Markup Language* é a linguagem de marcação utilizada na Web, utiliza tags para definição de elementos.
- **CSS3:** *Cascating Style Sheets* é utilizado para adicionar estilos ao HTML.

2.3 Manutenção de Software

Segundo Sommerville (2011), existem três tipos diferentes de manutenção de *software*:

- Manutenção para reparar falhas de *software*: Erros de codificação são geralmente relativamente baratos para corrigir; erros de projeto são mais caros, pois podem envolver a reescrita de vários componentes do programa. Erros de requisitos são os mais caros para reparar devido ao amplo redesenho do sistema que pode ser necessário.
- Manutenção para adaptar o *software* a um ambiente operacional diferente: Este tipo de manutenção é necessária quando algum aspecto do ambiente do sistema, como o hardware, o sistema operacional da plataforma ou outro *software* de suporte, muda. O sistema de aplicação deve ser modificado para adaptá-lo para lidar com essas mudanças ambientais.
- Manutenção para adicionar ou modificar a funcionalidade do sistema: Este tipo de manutenção é necessário quando os requisitos do sistema mudam em resposta a mudanças organizacionais ou de negócios. A escala das mudanças necessárias

ao software é muitas vezes muito maior do que para os outros tipos de manutenção.

De forma geral, esses tipos de manutenção são reconhecidos na literatura. A manutenção corretiva é universalmente usada para se referir à manutenção para correção de falhas. No entanto, a manutenção adaptativa às vezes significa adaptar-se a um novo ambiente e pode significar adaptar o *software* a novos requisitos. A **manutenção perfectiva** pode significar aperfeiçoar o *software* implementando novos requisitos; em outros casos, significa manter a funcionalidade do sistema, mas melhorar sua estrutura e seu desempenho (SOMMERVILLE, 2011).

2.3.1 Manutenção Perfectiva

A manutenção perfectiva consiste na manutenção de *software* realizada para melhorar o desempenho, manutenibilidade ou outros atributos de um programa de computador (ISO/IEC/IEEE 24765, 2017)

No entanto, a manutenção perfectiva não envolve somente a refatoração do código, atualizando as tecnologias utilizadas, mas também a manutenção estrutural do sistema, envolvendo desde estruturas de dados até a documentação, etc. (SOMMERVILLE, 2011). Essa manutenção estrutural pode ser chamada de reengenharia de *software*. O processo de reengenharia de *software* pode ser observado na Figura 3:

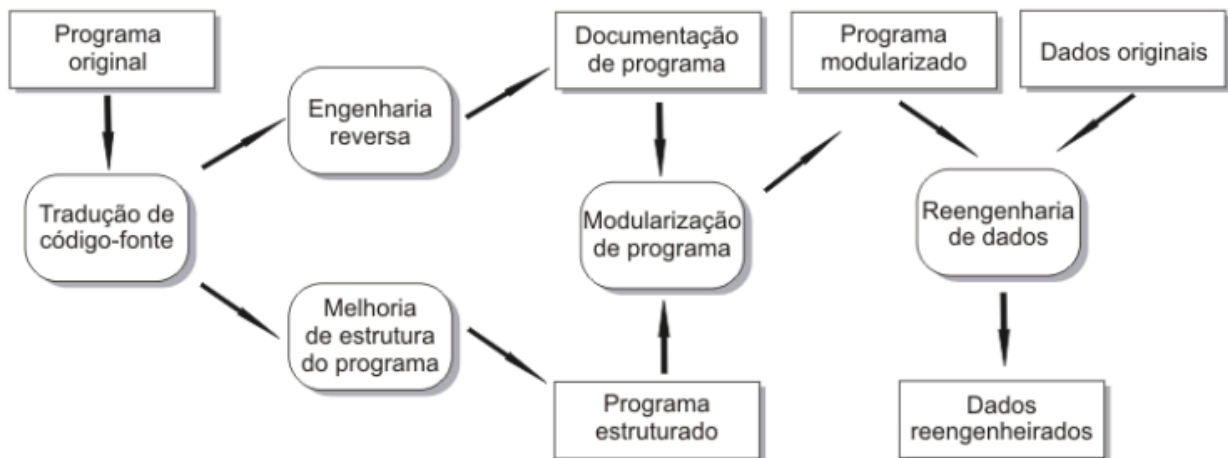


Figura 3 - Processo geral de reengenharia de software (SOMMERVILLE, 2011)

O processo de reengenharia de *software* tem como entrada o programa original que será refatorado e, como saída, sua versão reestruturada.

- Tradução do código-fonte: O programa é traduzido de uma linguagem de programação antiga, para a versão atualizada da mesma, ou para uma linguagem de programação diferente;
- Engenharia reversa: O programa é analisado para extrair mais informações, utilizando-as para ajudar na documentação e organização de seus casos de uso;
- Melhoria de estrutura do programa: A estrutura do programa é analisada e se necessário, alterada para torná-la mais fácil de ser compreendida;
- Modularização do programa: As partes do programa que tenham alguma relação são agrupadas e, em casos de agrupamentos redundantes, estes podem ser removidos;
- Reengenharia de dados: Atualização dos dados processados pelo *software*, para refletirem as alterações feitas no mesmo, como alterações de tabelas do banco de dados;

Não é obrigatório que todos os passos supracitados sejam realizados para que a reengenharia seja considerada bem sucedida.

De modo geral, utiliza-se o termo *legado* para se referir aos sistemas desenvolvidos há muito tempo com o uso de tecnologias e/ou metodologias defasadas, que atualmente são difíceis de modificar e/ou evoluir (FREITAS, 2017). Além da tecnologia defasada, fatores como dificuldade de compreensão do código, falta de documentação apropriada e estrutura degradada dificultam a manutenção do código (NILSSON, 2015; KANGASAO, 2016).

Alguns sintomas ligados ao envelhecimento do *software* são relacionados a poluição ocasionada por códigos desnecessários, conhecimento de negócio embutido na codificação, nomenclaturas inadequadas, acoplamentos indevidos e violações nas camadas da arquitetura (VISAGGIO, 2001).

Estes problemas levantados têm solução, segundo Comella-Dorda et al. (2000), a manutenção, modernização e substituição de sistemas de informação fazem parte das fases do seu ciclo de vida:

- Manutenção: Consiste em pequenas alterações que ocorrem com o passar do tempo, como correções e melhorias;
- Modernização: Consiste normalmente na reestruturação do sistema, sendo muito mais extensa que a manutenção;
- Substituição: Consiste na remoção do sistema, para utilização de um novo;

A modernização é uma evolução do sistema, simplificando a manutenção e facilitando a evolução com desenvolvimento de novas funcionalidades (Weiderman, 1997).

Segundo Bisbal et al. (1999), as técnicas de modernização podem ser divididas em 3 categorias:

- Redesenvolvimento: Refazer totalmente o sistema, despendendo-se de sua tecnologia defasada; esta categoria é a mais custosa;
- *Wrapping*: Desenvolver uma nova interface para parte do sistema ou para o sistema inteiro, permitindo novas integrações com outras interfaces;
- Migração: Migração incremental do sistema para outra plataforma;

A partir da fundamentação, podemos observar que a refatoração deste trabalho envolve, o redesenvolvimento do *software* se despendendo da tecnologia defasada, e pode-se considerar que também contempla o *Wrapping*, visto que a *API REST* poderá ser consumida por outras aplicações através da interface.

2.4 Refatoração

Refatorações são transformações parametrizadas do código-fonte de um sistema destinadas a melhorar a estrutura de um sistema em relação a objetivos expressos informalmente, como manutenibilidade, alterabilidade, legibilidade, desempenho ou demandas de memória. A refatoração geralmente preserva o comportamento do sistema (ARORA et al., 2011).

Segundo Fowler (1999) refatoração é o “processo de alterar um sistema de *software* de tal forma que não altere o comportamento externo do código, mas melhore sua estrutura interna”. Consiste em uma forma disciplinada de “limpar” o código, melhorando o design do código depois que ele foi escrito e minimizando as chances de introdução de bugs.

Métodos ágeis de desenvolvimento de *software*, como eXtreme Programming (XP) têm a refatoração como um de seus valores. A XP recomenda uma “refatoração impiedosa” para manter o design simples à medida que avança e evitar confusão e

complexidade desnecessárias. Programadores costumam manter seus projetos de *software* muito depois de terem se tornado difíceis de manejar, continuando a usar e reutilizar código que não é mais sustentável porque ainda funciona de alguma forma e têm medo de modificá-lo. A refatoração durante todo o ciclo de vida do projeto economiza tempo e aumenta a qualidade. (WELLS, 2022).

A refatoração dos componentes reutilizados geralmente é limitada pelo medo de quebrar o código do cliente. Quando mudanças de ruptura de uma API acontecem, ou seja, quando as interfaces dos componentes não são mais compatíveis, os desenvolvedores assumem o ônus de migrar seus códigos para a nova versão dos componentes reutilizados (ARORA et al., 2011).

Entretanto, uma série de pequenas refatorações podem levar a grandes mudanças, como a introdução do padrão de projeto. Um conjunto de mudanças pode levar a uma melhor compreensão da evolução do design de um sistema e a refatoração que ele sofreu (FOWLER, 1999).

Existem diversos tipos de refatoração (ARORA et al, 2011):

Refatoração de hierarquia de contenção: Os projetos são organizados em termos de subsistemas, pacotes e tipos de referência; tal organização torna as dependências entre os vários componentes explícitos e facilita a identificação do uso de um componente por seu contêiner implícito. Nesses casos, geralmente a hierarquia de contenção é reestruturada em diferentes níveis.

Refatoração de hierarquia de herança: Programar para interfaces e não para implementações é um princípio importante do desenvolvimento orientado a objetos. Quando o cliente é implementado para ser agnóstico da implementação interna da classe servidor, assumindo apenas a especificação de sua interface de comportamento público, o servidor mantém a flexibilidade para evoluir. Enquanto a interface pública permanecer

a mesma, as modificações em sua implementação não prejudicam seus clientes. Uma consequência do princípio de programação para interfaces é a refatoração típica de *Extract Interface*.

Refatoração de relacionamentos de classe: Sistemas orientados a objetos são basicamente projetados em torno de classes que modelam abstrações de entidades do mundo real e/ou encapsulamentos de um conjunto coerente de comportamentos.

Refatoração de classe interna: Vários tipos de refatoração reorganizam o código interno dentro de uma classe, incluindo generalização de tipo, introduzir padrão de projetos fábrica ou alterar assinatura de métodos.

São diversas as vantagens de se realizar refatoração (FOWLER, 1999):

A refatoração melhora o design de software: Sem refatoração, o design do programa decai ao longo do tempo. O código vai perdendo sua estrutura à medida que desenvolvedores modificam o código para atingir seus objetivos de curto prazo ou quando são realizadas mudanças sem uma compreensão completa do design do código. A perda da estrutura do código tem um efeito cumulativo, pois quanto mais difícil é ver o design no código, mais difícil é preservá-lo e mais rapidamente ele decai.

A refatoração torna o software mais fácil de entender: Muitas vezes quando um programa é desenvolvido não se pensa sobre futuros desenvolvedores que precisarão dar manutenção naquele código e, para isso, precisarão compreender o código. Compreender um código desorganizado pode agregar muito tempo à estimativa inicial de um desenvolvimento. Refatorar ajuda a melhorar a compreensibilidade do código;

Refatorar ajuda a encontrar bugs: Ao ajudar na compreensão do código, a refatoração também ajuda a identificar bugs ao esclarecer a estrutura do programa.

Refatoração ajuda a programar mais rápido: Inicialmente a refatoração pode aumentar o tempo de desenvolvimento, mas melhorar o design e melhorar a legibilidade é essencial para o desenvolvimento rápido de *software*, pois reduz o esforço necessário em futuras implementações.

Como a refatoração não deve mudar os requisitos funcionais do sistema, para se fazer uma refatoração os requisitos devem estar claramente identificados. Assim, um dos primeiros passos da refatoração consiste em construir um conjunto sólido de testes, a partir desses requisitos, para a seção de código que será alvo da refatoração. Esses testes são essenciais, pois à medida que a refatoração é realizada, é necessário confiar nos testes para que se possa dizer se foi introduzido algum bug (FOWLER, 1999). Assim, para este trabalho de conclusão de curso são utilizados os requisitos funcionais de todas as funcionalidades já introduzidas no sistema CodeMaster como base para os testes funcionais.

3 REFATORAÇÃO DO CODEMASTER

Neste capítulo é apresentada a refatoração do sistema CodeMaster, incluindo a análise da arquitetura atual, as propostas de melhoria e a implementação dessas melhorias.

3.1 ANÁLISE DA ARQUITETURA ATUAL

O CodeMaster foi desenvolvido usando a linguagem de programação *Java* com *JSP* (linguagem *Java* para desenvolvimento Web). Esta tecnologia escolhida pelo fato do aluno e equipe de desenvolvimento ter conhecimento prévio, e por serem as tecnologias disponíveis na infraestrutura de servidor utilizado (DEMETRIO, 2017).

Com o objetivo de tornar a arquitetura escalável, o modelo de arquitetura do sistema foi desenhado separando as camadas de apresentação e análise em módulos distintos. A arquitetura do sistema CodeMaster V1.0 é apresentada na Figura 4.

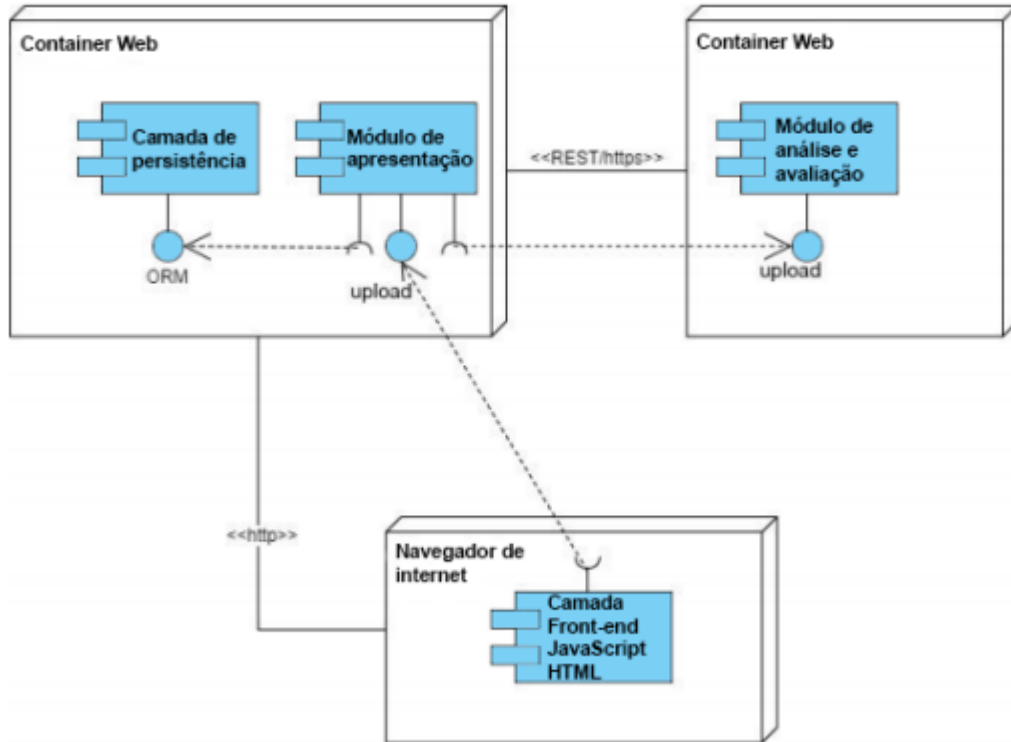


Figura 4 - Diagrama de componentes (PELLE, 2018)

O módulo de avaliação é um serviço web REST. Este serviço é responsável por receber o projeto, suas configurações para avaliação e retornar o resultado da avaliação (DEMETRIO, 2017).

O módulo de apresentação é responsável pela interface de usuário, login de professores, registro de professores e turmas, submissão de projetos, apresentação dos resultados (DEMETRIO, 2017), registros e logins de administradores e geração de relatórios (JUSTEN, 2019).

As tecnologias atualmente utilizadas no sistema CodeMaster são apresentadas no Quadro 1.

Recurso	Descrição
Java	JavaServer Pages (linguagem Java para desenvolvimento Web).
REST	Representational State Transfer.
JavaScript	É uma linguagem de programação interpretada de alto nível, dinâmica, fracamente tipada, prototype-based e multi-paradigma.
HTML5	Hypertext Markup Language, versão 5.
CSS3	Cascading Style Sheets, versão 3.
MySQL 5.5	Banco de dados relacional open source.

Quadro 1 - Tecnologias utilizadas no CodeMaster (DEMETRIO, 2017)

A implementação para a evolução do CodeMaster, como um processo de manutenção perfectiva de reengenharia, deve evoluir a arquitetura atual do *software*, melhorando a separação entre camadas de apresentação e análise em diferentes módulos.

Atualmente a camada de apresentação é responsável pelo controle de interface do usuário, essa camada recebe o projeto do usuário por meio da interface web, envia o projeto para o analisador e apresenta o resultado da avaliação ao usuário, além de ser responsável por toda a parte de persistência de dados do sistema.

O módulo de análise tem atualmente a responsabilidade de receber o projeto, as configurações e retornar o resultado da avaliação.

4.1 ANÁLISE DAS PRINCIPAIS NECESSIDADES DE MANUTENÇÃO

Com a evolução do sistema CodeMaster ao longo do tempo, diversas necessidades de manutenção foram percebidas pelos desenvolvedores e pesquisadores que interagiram com o seu código-fonte. Essas necessidades são discutidas a seguir.

4.1.1 Modernização das tecnologias:

A arquitetura e algumas tecnologias empregadas no CodeMaster V1.0 são consideradas defasadas, como *Servlets* e *JSP*, gerando grandes dificuldades para a evolução da ferramenta devido à falta de desenvolvedores aptos e com conhecimento nestas tecnologias.

A completa separação das camadas *front-end* e *back-end* melhoram a manutenibilidade do código, sendo que novos desenvolvedores não necessariamente precisam ter conhecimento em ambas as camadas. Com a refatoração, foram utilizadas tecnologias atuais e com grande aceitação no mercado, como o *SpringBoot* e *Angular*, o que pode ser considerado como um atrativo para que novos desenvolvedores possam propor novas funcionalidades para evoluir a aplicação.

4.1.2 Separação de camadas

A separação do *software* em camadas de *front-end* e *back-end* é uma estratégia muito utilizada no desenvolvimento de *software* por vários motivos, como simplificar o desenvolvimento e a manutenção da aplicação e a possibilidade de serem desenvolvidos simultaneamente por diferentes equipes. O fato de possuir códigos separados e independentes de tecnologia permite a alteração de uma das tecnologias sem que haja

a necessidade de alterar a outra camada, além da possibilidade de hospedar o *front-end* em um servidor e o *back-end* em outro servidor.

Outro ponto importante é a separação das camadas lógicas dentro de cada projeto, o que permite o agrupamento de componentes similares, aumentando o suporte a reutilização e coesão, além de reduzir o acoplamento do sistema. Esta divisão também proporciona facilidade para manutenção, tornando o código mais intuitivo para o desenvolvedor.

Desta forma a aplicação é separada em 3 camadas lógicas:

- Camada de Apresentação: É exposta aos clientes através de uma *API REST* que recebe as requisições externas;
- Camada de Negócios: É a camada que possui domínio das regras de negócios, interligando as solicitações dos usuários com a base de dados, de acordo com as regras;
- Camada de Dados: É responsável pelo acesso e persistência dos dados do sistema, geralmente são utilizados *frameworks* como o *Hibernate* para estes acessos, vale ressaltar que essa camada não deve possuir nenhuma regra de negócio;

No entanto, a forma como a atual arquitetura do CodeMaster V1.0 utiliza *Java Server Pages*, tornou inviável colocar em prática essa separação em camadas. No código atual, tem-se as classes *JSP* que utilizam várias tecnologias simultaneamente além do *Java*, como o *HTML*, linguagem de marcação utilizada para criação de páginas na *Web*, o *CSS*, utilizado para estilizar os elementos do *HTML*, o próprio *JavaScript*, juntamente com as importações de bibliotecas e outras classes *JSP*. Como exemplo, o código *JSP* da classe Aluno apresentando o resultado de avaliação de código Snap! do sistema atual, conforme Figura 5.

```

<head>
  <style>
    table {
      width: 100%;
      border: 1px solid #cccccc;
    }
  </style>
  <meta charset="utf-8"><link rel="stylesheet" href="css/styles.css">
</head>
<body>
  <%Gson gson = new Gson();
  String msg = request.getAttribute("msg").toString();
  JsonParser jsonParser = new JsonParser();
  JSONArray arrayGrades = (JSONArray)jsonParser.parse(msg);
  SnapGrade g = gson.fromJson(arrayGrades.get(0), SnapGrade.class);
  NumberFormat nf = NumberFormat.getInstance();
  nf.setMaximumFractionDigits(1);
  nf.setMinimumFractionDigits(1);
  String level = g.getProjectLevel();
  String displayLevel = "";
  switch(level) {
    case "faixa branca": displayLevel = messages.getString("FaixaBranca"); break;
    case "faixa amarela": displayLevel = messages.getString("FaixaAmarela"); break;
    case "faixa laranja": displayLevel = messages.getString("FaixaLaranja"); break;
    case "faixa vermelha": displayLevel = messages.getString("FaixaVermelha"); break;
    case "faixa roxa": displayLevel = messages.getString("FaixaRoxa"); break;
    case "faixa azul": displayLevel = messages.getString("FaixaAzul"); break;
    case "faixa turquesa": displayLevel = messages.getString("FaixaTurquesa"); break;
    case "faixa verde": displayLevel = messages.getString("FaixaVerde"); break;
    case "faixa marrom": displayLevel = messages.getString("FaixaMarrom"); break;
    case "faixa preta": displayLevel = messages.getString("FaixaPreta"); break;
  }%>
  <table class="table"><thead class="">
    <tr class="table-active">
      <th class="text-center"><%=messages.getString("ColunaConceito")%></th>
      <th class="text-center"><%=messages.getString("ColunaPontuacao")%></th>
    </tr></thead>
    <tbody>
      <tr>
        <td><%=messages.getString("ConceitoLogica")%></td>
        <td><div class="progress">
          <div class="progress-bar progress-bar-striped"
            <%=getProgressBar(g.getLogic())%>
            role="progressbar" aria-valuenow="<%=g.getLogic()%>"
            aria-valuemin="0" aria-valuemax="3"
            style="width: <%= (100.0 / 3) * g.getLogic()%>%">
            <span><strong class="text-gray-dark">
              <%=g.getLogic()%>/3</strong></span>
          </div>
        </div></td>
      </tr>
    </tbody>
  </table>
  <script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>
  <script src="https://pingendo.com/assets/bootstrap/bootstrap-4.0.0-alpha.6.min.js"></script>
</body>

```

Figura 5 - Código do resultado de avaliação do código Snap!

É possível perceber no código da Figura 5 que o código *Java* “contamina” a camada de apresentação, adicionando regras de negócio que deveriam estar do lado do servidor. Com isso, a manutenção do código fica prejudicada. Por exemplo, se um novo critério de avaliação ou se uma nova “faixa” precisar ser adicionada como uma nova funcionalidade ou como resultado de uma alteração em alguma regra de negócio, além do código do lado do servidor, o código da interface também seria bastante impactado. Como esse tipo de situação está espalhado por todo o sistema, quaisquer alterações ou acréscimos de funcionalidade são dificultados, podendo causar bugs pela alteração ser realizada no código do lado do servidor e não do lado da apresentação, ou vice-versa.

Além disso, há diversos problemas no uso do *CSS* na camada de apresentação. O arquivo *style.css* do sistema possui atualmente mais de 6 mil linhas, e a maioria das classes *JSP* do sistema adicionam outras configurações próprias de *CSS*. Esse fato aumenta muito a dificuldade de manutenção para a incorporação de novas funcionalidades ou até mesmo a realização de pequenos ajustes de formatação visual dos componentes.

4.1.3 Inclusão de diversas novas funcionalidades ao longo do tempo

Com a evolução da pesquisa na área de avaliação de código realizada pelo grupo de pesquisas GQS, várias novas funcionalidades foram sendo incorporadas ao sistema inicial do CodeMaster desenvolvido por Demetrio (2017).

A primeira funcionalidade incorporada no CodeMaster (DEMETRIO, 2017), foi a análise e avaliação de aplicativos desenvolvidos na linguagem Snap! (PELLE, 2018). A Figura 6 apresenta a tela de resultado da avaliação de código Snap!.

Avaliação de projeto Snap!



Figura 6 - Tela de resultado da avaliação de projeto Snap!

Outras funcionalidades adicionadas foram a análise e avaliação automática de design de interface de aplicativos desenvolvidos com o App Inventor em relação a sua conformidade com diretrizes de design de interface (JUSTEN, 2019), conforme a Figura 7. Essas novas funcionalidades aumentaram a complexidade da camada de apresentação, pois foram criadas novas áreas de visualização de um volume considerável de dados. Com isso, as páginas *JSP* ficaram ainda mais difíceis de manter.

Avaliação de projeto App Inventor

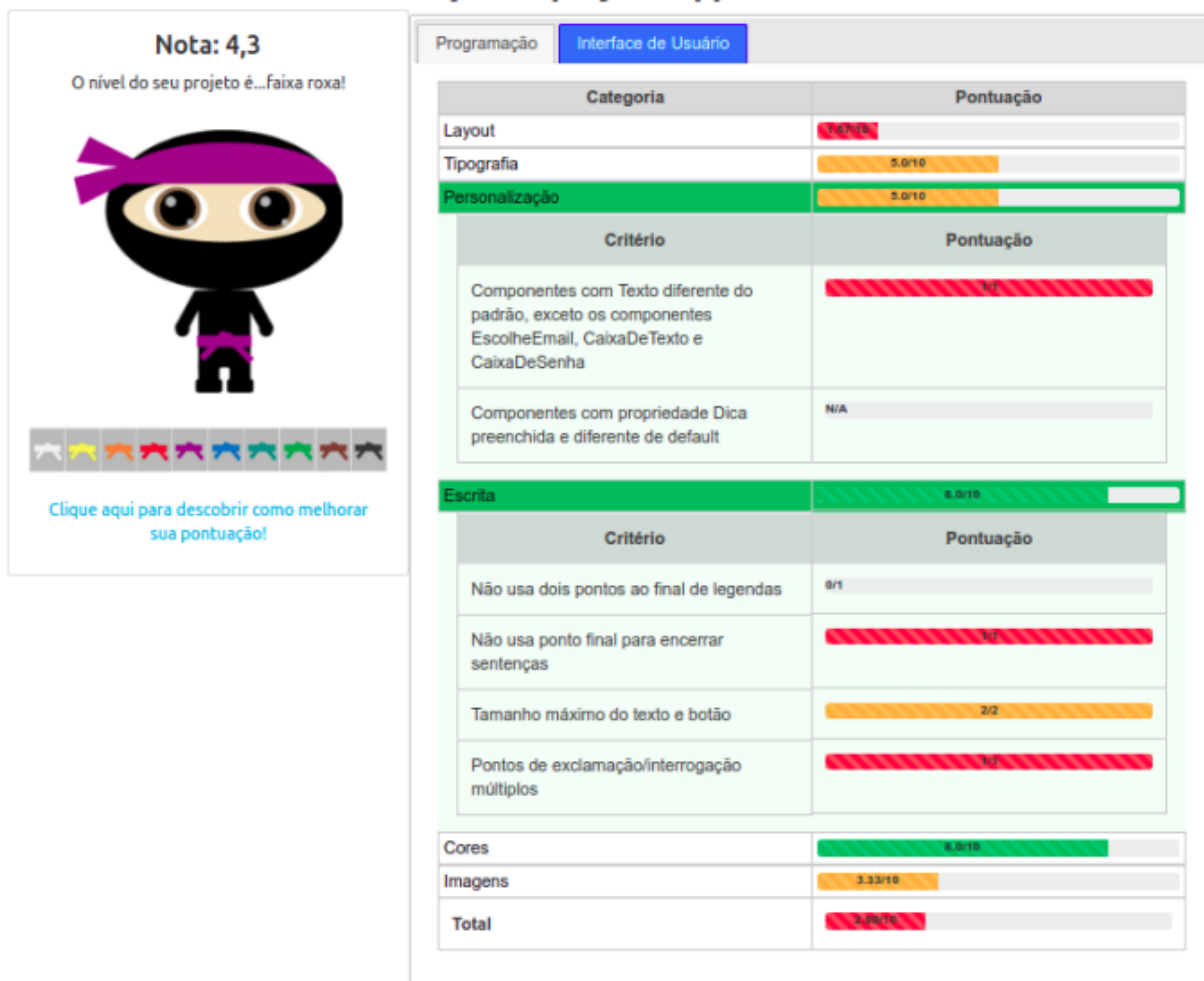


Figura 7 - Tela de resultado da avaliação de Interface Gráfica de projeto App Inventor.

Foi adicionado também uma área de login e cadastro para Administradores exportarem relatórios de informações salvas na base de dados da aplicação (JUSTEN, 2019), conforme mostrado nas Figuras 8 e 9. Essa nova funcionalidade agregou a geração de arquivos .xls com os resultados da consulta e a possibilidade de download, alterando as rotas da aplicação utilizadas até então. Além disso, o código de geração do xls utiliza componentes antigos e de código bastante prolixo.

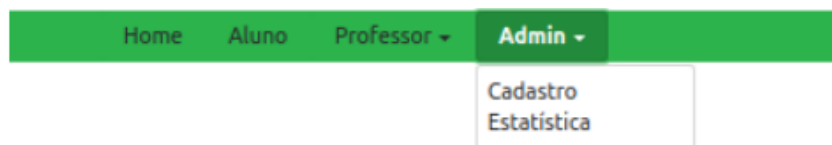


Figura 8 - Cadastro e Estatísticas para Pesquisadores

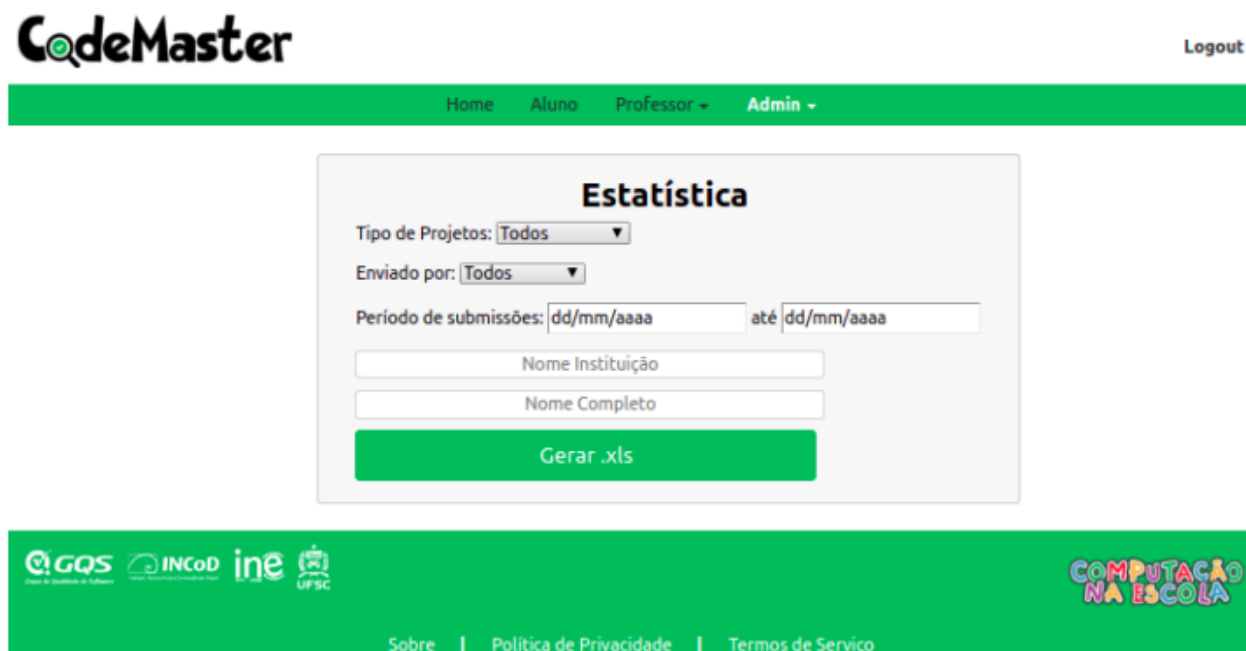


Figura 9 - Consulta para Pesquisadores

Outra funcionalidade incorporada no CodeMaster foi a avaliação estética de imagens de telas de aplicativos, que ocorre por meio de um mini-servidor *Python* acessado pela aplicação (MARTINS, 2019). Especificamente esta funcionalidade agregou outras tecnologias à ferramenta, como o uso de *Python* e bibliotecas de *machine learning*. Para essa integração foi adaptada a comunicação entre os módulos de apresentação (*JSP*) e um mini-servidor em *Python*, tornando ainda mais complexa e de difícil manutenção essa parte do código.

Home Aluno Professor Admin

Avaliação de projeto App Inventor

Nota: 0.0
O nível do seu projeto é...faixa branca!

Clique aqui para descobrir como melhorar sua pontuação!

Categoria	Pontuação
Layout	0.0/10
Tipografia	0.0/10
Escrita	0.0/10
Cores	0.0/10
Imagens	0.0/10
Total	0.0/10

Avaliação Estética

Pontuação

10/10

Choose Files 1483_rico.jpg

Figura 10 - Área de estética na interface de usuário.

Prevê-se também a inclusão de novos módulos enfocando na avaliação de novas dimensões de projetos de App Inventor. Além de novos módulos voltados a avaliação automatizada também de artefatos relacionado a aprendizagem de Machine Learning.

4.2 Seleção de novas tecnologias

4.2.1 Critérios de seleção

A partir da identificação da necessidade de atualização das tecnologias utilizadas no desenvolvimento do CodeMaster, foram levantados os principais critérios de seleção para as tecnologias:

- Deve-se manter o *back-end* na linguagem *Java*, para que não seja necessário reimplementar todo o sistema. Além disso, a linguagem *Java* é uma linguagem amplamente conhecida, e ainda há facilidade de encontrar programadores com conhecimento, facilitando futuras manutenções;
- Tecnologias atuais e com comunidades ativas;
- Preferencialmente, o autor ter conhecimento prévio na tecnologia.

4.2.2 Seleção das novas tecnologias

Para as tecnologias do *back-end*, sendo um dos critérios de seleção a manutenção da linguagem *Java*, foi utilizado o *framework Spring Boot* (<https://start.spring.io/>), com o qual o autor deste trabalho é familiarizado. Este *framework*, além de ser atual e ter uma comunidade ativa, agrega diversas *features* à aplicação, como bibliotecas de segurança, simplicidade de acesso a base de dados e a camada *REST* para comunicação com outras interfaces.

Para o *front-end*, foi escolhido o *framework Angular*, versão 12, lançado em maio de 2021, versão mais atual na data do início da refatoração (<https://angular.io/>). A versão 13 foi lançada recentemente e, como as versões anteriores, é retrocompatível. O *Angular* é baseado em *TypeScript*, com o código-fonte aberto e com a equipe de desenvolvimento sendo liderada pelo *Google*. O autor deste trabalho não tinha experiência com essa versão específica do *Angular*, mas possuía experiência com versões anteriores.

4.3 IMPLEMENTAÇÃO DA MANUTENÇÃO

4.3.1 Configuração do ambiente de desenvolvimento

Para o desenvolvimento deste trabalho, foram utilizados a *IDE Eclipse*¹ para o *back-end* em *Java*. Como não houve alterações na base de dados, a refatoração manteve o uso do *MySQL 5.5*, e do *TomCat 8*, que já eram utilizados na versão do sistema legado.

No *front-end* foi utilizado a *IDE Visual Studio Code*², para o desenvolvimento do código *Angular 12*. Também é necessário a instalação do *node.js*³ para o desenvolvimento do *Angular*.

O manual de instalação do ambiente está em desenvolvimento e será publicado no repositório do projeto junto com o código-fonte e na wiki do GQS.

4.3.2 Principais dificuldades da manutenção de código

Durante o processo de refatoração surgiram diversas dificuldades, sendo que algumas já eram esperadas. Uma das primeiras dificuldades encontradas se refere à

¹ <https://www.eclipse.org/>

² <https://code.visualstudio.com/>

³ <https://nodejs.org/en/>

curva de aprendizado do *Angular* 12. Conforme já citado, o autor possuía experiência com versões anteriores do *Angular*, mas não com essa versão específica, o que ocasionou dificuldades desde a criação do projeto, primeiras configurações e desenvolvimento dos primeiros artefatos. O uso do *TypeScript*, também representou algumas dificuldades iniciais.

Outro ponto de grande dificuldade refere-se à implementação do *front-end*. Como este projeto se trata de uma manutenção perfectiva de reengenharia de *software*, as funcionalidades já existentes não deveriam ser impactadas pela implementação da manutenção, incluindo o *layout* e estética das telas do sistema. Assim, uma das maiores dificuldades foi manter o mesmo *layout* do sistema antigo, pois, como já citado, o arquivo *style.css* do sistema legado possui mais de 6 mil linhas, e a maioria das classes *JSP* do sistema legado tinham outras configurações próprias de *CSS*.

Em relação à manutenção do código-fonte do *back-end*, a refatoração representou menores dificuldades. Com a utilização das bibliotecas disponibilizadas pelo *Spring Boot*, foi possível simplificar boa parte do código, tornando-o mais objetivo e intuitivo para futuras manutenções e evoluções.

Outra importante dificuldade se deu em sincronizar a implementação da refatoração enquanto outros desenvolvedores continuavam realizando desenvolvimentos de novas funcionalidades no sistema CodeMaster. Foram realizadas duas etapas de *merge* com outro desenvolvedor que estava realizando desenvolvimento. Numa primeira etapa, quando uma primeira parte da refatoração já havia sido realizada e um segundo *merge* quando a parte final da refatoração já havia sido implementada. No entanto, isso gerou retrabalho para o outro desenvolvedor, pois entre uma etapa e outra, diversas alterações foram feitas no *CSS*, implicando na necessidade de reconfigurar as telas que já haviam sido desenvolvidas.

4.3.2 Detalhamento das modificações

Nesta seção é apresentada a arquitetura do *CodeMaster V2.0*.

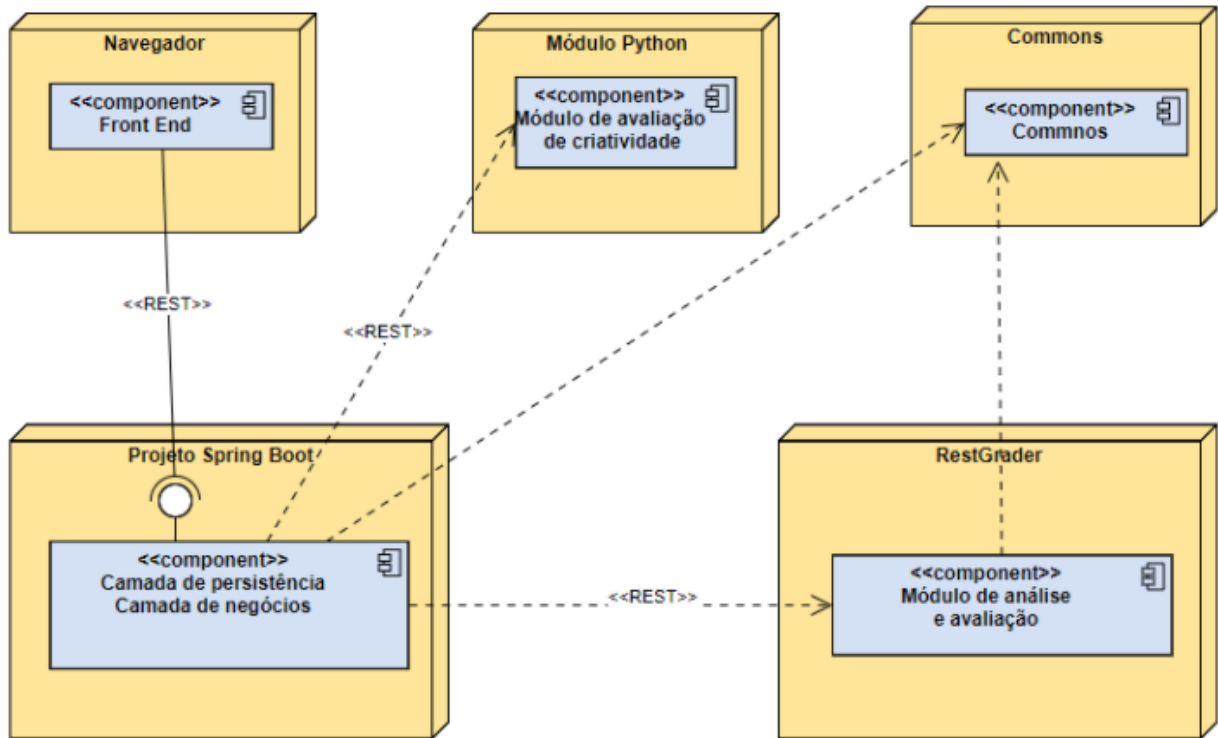


Figura 11 - Arquitetura do CodeMaster V2.0

O módulo de análise e avaliação, denominado *RESTGrader* e o *Commons* não tiveram alterações significativas, pequenas correções para padronização. O módulo de apresentação do *CodeMaster V1.0* foi dividido em dois projetos distintos, separando *front-end* de *back-end*. O *front-end* é responsável pela interface com o usuário, recebendo parâmetros e apresentando os resultados das avaliações que são realizadas no *back-end*. O *back-end* é responsável pela camada de negócios, persistência de dados e interação com os módulos de análise.

Abaixo são apresentados alguns exemplos de melhoria estrutural em diversas partes do código-fonte. São apresentados trechos de código antes da manutenção e

depois da manutenção.

A nova versão do código-fonte do CodeMaster está disponível em: <https://codigos.ufsc.br/100000000394729/CodeMaster>.

O código apresentado na Figura 12 representa o menu principal do sistema, responsável por toda navegação da aplicação, é um exemplo claro de não reaproveitamento de código, pois este menu é replicado em 21 classes da atual implementação.

```
<div class="bg-success">
  <div class="container">
    <div class="row">
      <div class="col-centered">
        <div class="btn-group">
          <a href="index.jsp" class="btn btn-success text-gray-dark"><%=messages.getString("HomeBtn")%></a>
          <a href="aluno.jsp" class="btn btn-success text-gray-dark"><%=messages.getString("AlunoBtn")%></a>
          <div class="dropdown">
            <button class="btn btn-success dropdown-toggle" type="button"
              data-toggle="dropdown">
              <b><%=messages.getString("ProfessorBtn")%></b><span
                class="caret"></span>
            </button>
            <ul class="dropdown-menu text-left px-2">
              <li><a href="professor.jsp" class="text-gray-dark"><%=messages.getString("ProfessorDropAvaliar")%></a></li>
              <li class="divider"></li>
              <li><a href="professor_turmas.jsp" class="text-gray-dark"><%=messages.getString("ProfessorDropTurmas")%></a></li>
            </ul>
          </div>
          <div class="dropdown">
            <button class="btn btn-success dropdown-toggle" type="button"
              data-toggle="dropdown">
              <b><%=messages.getString("AdminBtn")%></b><span class="caret"></span>
            </button>
            <ul class="dropdown-menu text-left px-2">
              <li><a href="admin.jsp" class="text-gray-dark"><%=messages.getString("AdministradorDropCadastro")%></a></li>
              <li class="divider"></li>
              <li><a href="admin_estatic.jsp" class="text-gray-dark"><%=messages.getString("AdministradorDropEstatistica")%></a></li>
            </ul>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Figura 11 - Código do menu do CodeMaster legado.

Na refatoração, foi criado um componente chamado *Header*, que é responsável pelo menu do sistema, simplificando qualquer processo de manutenção ou

implementação de novas funcionalidades, como por exemplo a criação do menu “Machine Learning para Todos” que está sendo desenvolvido no código refatorado em paralelo com este trabalho (LAYDNER, 2022). No CodeMaster V1.0 (DEMETRIO, 2017), este menu seria adicionado em 21 classes diferentes, no código refatorado, é adicionado apenas no componente header.component.html e header.component.ts conforme Figuras 13 e 14.

```
1 <header class="main-header">
2   <nav class="navbar navbar-static-top">
3     <div class="container">
4       <div class="col-md-10">
5         
6       </div>
7       <div class="col-md-2">
8         <a href="/logout" class="text-gray-dark">Logout</a>
9       </div>
10    </div>
11    <div class="col-md-12">
12      <div class="bg-success col-md-12">
13        <div class="col-md-3"> </div>
14        <div class="container row col-md-6">
15          <div class="wrapper text-center">
16            <div class="btn-group" style="text-align: center;">
17              <button class="btn btn-success text-center bg-success text-gray-dark"
18                (click)="goHome()">Home</button>
19              <button class="btn btn-success text-center bg-success text-gray-dark"
20                (click)="goAluno()">Aluno</button>
21              <button class="btn btn-success text-center bg-success text-gray-dark"
22                (click)="goProfessor()">Professor</button>
23              <button class="btn btn-success text-center bg-success text-gray-dark"
24                (click)="goAdministrador()">Administrador</button>
25              <button class="btn btn-success text-center bg-success text-gray-dark"
26                (click)="goMLPT()">Machine Learning para Todos</button>
27            </div>
28          </div>
29        </div>
30        <div class="col-md-3"> </div>
31      </div>
32    </div>
33  </nav>
34 </header>
```

Figura 12 - Novo HTML do menu do CodeMaster V2.0.

```

1  import { Component, OnInit } from '@angular/core';
2  import { Router } from '@angular/router';
3
4  @Component({
5    selector: 'cm-header',
6    templateUrl: './header.component.html',
7    styleUrls: ['./header.component.css']
8  })
9  export class HeaderComponent implements OnInit {
10
11    constructor(
12      private router: Router
13    ) { }
14
15    ngOnInit(): void {
16    }
17
18    goHome() {
19      this.router.navigate(['/']);
20    }
21
22    goAluno() {
23      this.router.navigate(['/aluno']);
24    }
25
26    goProfessor() {
27      this.router.navigate(['/professor']);
28    }
29
30    goAdministrador() {
31      this.router.navigate(['/administrador']);
32    }
33
34    goMLPT() {
35      this.router.navigate(['/machinelearningparatodos']);
36    }
37
38  }

```

Figura 13 - Novo TypeScript do menu do CodeMaster V2.0.

O mesmo acontecia com o rodapé da aplicação. Cujo código era replicado em 21 classes *JSP*. Agora a aplicação tem 1 único componente *Footer* sendo utilizado em todo o sistema.

Outras alterações simplificaram bastante a refatoração do *back-end*. Em seguida veremos alguns exemplos da camada de acesso a base de dados. A Figura 15 representa a consulta das turmas de um professor no sistema legado.

```
public List<Turma> buscaTurmas(Integer id_professor){
    try {
        Connection conexao = FabricaConexao.getConexao();
        PreparedStatement ps = conexao.prepareStatement("SELECT * "
            + "FROM bd_code_master.turma WHERE id_professor = ?;");
        ps.setInt(1, id_professor);
        ResultSet resultSet = ps.executeQuery();
        List<Turma> turmas = new ArrayList<>();
        while(resultSet.next()){
            Turma turma = new Turma();
            turma.setId(resultSet.getInt("id_turma"));
            turma.setNome(resultSet.getString("nome"));
            turmas.add(turma);
        }
        return turmas;
    } catch (SQLException ex) {
        Log.error(ex);
        return null;
    }
}
```

Figura 14 - Consulta de turmas de professor no CodeMaster legado.

Já a Figura 16 representa a mesma consulta após a refatoração, simplificada com a utilização do *Java Persistence API*.


```
@Query(" SELECT p FROM Turma p WHERE "  
+ "p.professor.identificador = :idProfessor ")  
public List<Turma> getTurmasProfessor(  
    @Param(value = "idProfessor") Integer idProfessor);
```

Figura 15 - Nova consulta de turmas de professor no CodeMaster.

Outros exemplos de simplificação da camada de acesso aos dados estão nos exemplos a seguir. Consultas pelo identificador dos objetos já são implementadas pelo *JPA* nativo, conforme apresentado na Figura 17.

```
Turma turma = turmaRepository.findById(idTurma).get();
```

Figura 16 - Nova consulta por identificador.

A persistência de dados no sistema legado também era custosa. Era necessário criar a inserção na base de dados através da criação de *SQL*, conforme apresentado na Figura 18.

```

public void salvar(Turma turma) {
    try {
        Connection conexao = FabricaConexao.getConexao();
        PreparedStatement ps = conexao.prepareStatement(
            "INSERT INTO `bd_code_master`.`turma` (`nome`, `id_professor`) VALUES (?, ?);",
            Statement.RETURN_GENERATED_KEYS
        );
        ps.setString(1, turma.getNome());
        ps.setInt(2, turma.getIdProfessor());

        ps.execute();

        final ResultSet rs = ps.getGeneratedKeys();
        if (rs.next()) {
            final int ultimoId = rs.getInt(1);
            turma.setId(ultimoId);
        }
        FabricaConexao.fecharConexao();
    } catch (SQLException ex) {
        log.error(ex);
    }
}

```

Figura 17 - Insert de Turma no CodeMaster legado.

Para persistir um objeto na base de dados também é utilizada a interface genérica do *JPA*, bastando apenas criar a instância do objeto desejado e chamar a execução do repositório através do método nativo `save`, conforme as Figuras 19 e 20 respectivamente.

```

Turma turma = new Turma();
turma.setNome(nomeTurma);
turma.setProfessor(new Professor(professor.getIdificador()));
turma = turmaService.cadastrarTurma(turma);

```

Figura 18 - Criação da instância da Turma no CodeMaster.

```

@Override
public Turma cadastrarTurma(Turma turma) {
    return turmaRepository.save(turma);
}

```

Figura 19 - Insert da instância da Turma na base de dados do CodeMaster.

Estas facilidades disponibilizadas em *frameworks* mais atuais, impactam significativamente a produtividade do desenvolvedor, além de simplificar o código-fonte, facilitando a manutenibilidade de outros desenvolvedores.

5 TESTES FUNCIONAIS

Um dos principais critérios para a refatoração é que as modificações internas do código não devem gerar alterações perceptíveis no comportamento externo do *software*. Assim, foram levantados todos os requisitos dos trabalhos anteriores de desenvolvimento de funcionalidades do CodeMaster (DEMETRIO, 2017; PELLE, 2018; JUSTEN, 2019; MARTINS, 2019) e foram realizados testes funcionais para avaliar o comportamento do sistema para cada um dos requisitos.

5.1 Requisitos Funcionais do Codemaster

No Quadro 2 são apresentados os requisitos originais do CodeMaster e das suas posteriores evoluções, conforme relatado nos documentos de trabalho de conclusão de curso onde essas implementações foram devidamente documentadas.

ID	Requisito	Descrição	Fonte
RF01	Realizar Upload de um projeto App Inventor	A ferramenta deve permitir que qualquer usuário envie um projeto App Inventor no formato .aia.	DEMETRIO, 2017
RF02	Realizar upload de um projeto Snap!	A ferramenta deve permitir que qualquer usuário realize upload de um projeto Snap! no formato XML via web.	PELLE, 2018
RF03	Avaliar o design visual do Projeto App Inventor	A ferramenta deve avaliar cada critério e apresentar o nível de desempenho alcançado em cada critério	JUSTEN, 2019
RF04	Realizar Upload de um conjunto de projetos App Inventor	A ferramenta deve permitir que um professor cadastrado envie um conjunto de projetos (o qual representa uma turma) App Inventor no formato .aia.	DEMETRIO, 2017
RF05	Realizar upload de conjunto de projetos Snap!	A ferramenta deve permitir que um usuário cadastrado realize upload de um conjunto de projetos Snap! no formato XML via web. O autor de cada projeto é identificado por meio do nome do arquivo.	PELLE, 2018
RF06	Analisar Projeto App Inventor	A ferramenta deve realizar a análise quantitativa dos elementos definidos para cada conceito.	DEMETRIO, 2017
RF07	Analisar Projeto Snap!	A ferramenta deve ser capaz de analisar o código de	PELLE, 2018

		um projeto Snap! identificando a frequência de uso de cada bloco programado no projeto.	
RF08	Avaliar Projeto App Inventor	A ferramenta deve avaliar cada conceito de acordo com os elementos especificados encontrados no projeto App Inventor e gerar uma nota para cada conceito.	DEMETRIO, 2017
RF09	Avaliar projeto Snap!	A ferramenta deve ser capaz de avaliar os conceitos de computação, a partir da frequência de uso de blocos seguindo rubrica.	PELLE, 2018
RF10	Apresentar Avaliação do projeto App Inventor de forma detalhada	A ferramenta deve apresentar na interface de usuário a avaliação detalhada do seu projeto. Deve apresentar a nota de cada conceito separadamente, bem como a nota final e o nível de competência do aluno e o feedback	DEMETRIO, 2017
RF11	Apresentar avaliação detalhada do projeto Snap!	A ferramenta deve ser capaz de apresentar a avaliação feita em sua interface com o usuário de forma detalhada.	PELLE, 2018
RF12	Apresentar Avaliação dos projetos App Inventor de forma resumida	A ferramenta deve apresentar na interface de usuário a avaliação de um conjunto de projetos de forma resumida ao professor.	DEMETRIO, 2017
RF13	Apresentar avaliação resumida dos projetos Snap!	A ferramenta deve ser capaz de apresentar múltiplas avaliações feitas em sua interface com o usuário de forma resumida.	PELLE, 2018
RF14	Realizar Cadastro de Professor	A ferramenta deve permitir o cadastro de professores por meio de sua interface Web.	DEMETRIO, 2017
RF15	Realizar Login de Professor	A ferramenta deve permitir que o professor autentique sua identidade por meio de login.	DEMETRIO, 2017
RF16	Realizar login de Administrador	A ferramenta deve permitir que o administrador autentique sua identidade por meio de login.	JUSTEN, 2019
RF17	Realizar Logout de Professor	A ferramenta deve permitir que o professor encerre a sessão atual de autenticação de sua identidade por meio de logout.	DEMETRIO, 2017
RF18	Realizar Logout de administrador	A ferramenta deve permitir que o administrador encerre a sessão atual de autenticação de sua identidade por meio de logout.	JUSTEN, 2019
RF19	Manter Registro de Avaliações do Professor	Login do professor e projetos submetidos na sessão atual.	DEMETRIO, 2017
RF20	Manter Registro de Avaliação de Usuário Anônimo	A ferramenta deve manter em seu banco de dados o registro de uma avaliação feita por usuário anônimo.	DEMETRIO, 2017

RF21	Apresentar todas Avaliações do Professor	A ferramenta deve apresentar em sua interface com o usuário todas as avaliações já feitas pelo professor.	DEMETRIO, 2017
RF22	Apresentar Estatísticas de Avaliações	A ferramenta deve apresentar em sua interface com o usuário estatísticas sobre todos os projetos App Inventor já analisados.	DEMETRIO, 2017

Quadro 2 - Requisitos Funcionais do CodeMaster e suas evoluções.

5.2 Testes dos Requisitos Funcionais

Para realização dos testes funcionais, após a finalização das implementações de refatoração, o sistema CodeMaster foi colocado em execução no computador do autor deste trabalho e cada um dos requisitos foi testado em detalhes. A seguir para cada requisito são apresentadas evidências de resultado dos testes.

Como o passo a passo para testar alguns requisitos funcionais são iguais, alguns requisitos foram agrupados para eliminar a redundância de imagens.

5.2.1 Teste de funcionalidade do menu Aluno - App Inventor

Foram agrupados os seguintes requisitos:

RF01 - Realizar *Upload* de um projeto App Inventor

RF03 - Avaliar o *design* visual do Projeto App Inventor

RF06 - Analisar Projeto App Inventor

RF08 - Avaliar Projeto App Inventor

RF10 - Apresentar Avaliação do projeto App Inventor de forma detalhada.

No menu Aluno, o usuário deverá fazer o upload do arquivo App Inventor, selecionando o arquivo.aia que deseja avaliar, e clicar no botão avaliar, conforme Figura 21.

Avalie seu projeto

- 1) Clique no botão "Escolher arquivo" e selecione o projeto
 - Arquivo .xml para projeto Snap!
 - Arquivo .aia para projeto App Inventor
 - Arquivo .jpg para avaliação estética da interface de usuário (máximo 1024 pixels de altura e orientação tipo retrato)

MathExperienceaaaa.aia

- 2) Clique no botão "Avaliar".



Figura 20 - Upload de projeto App Inventor no CodeMaster.

O usuário será redirecionado para tela de resultado, que terá a aba Programação e Interface de Usuário, com as pontuações, nota final e faixa, conforme Figuras 22 e 23.

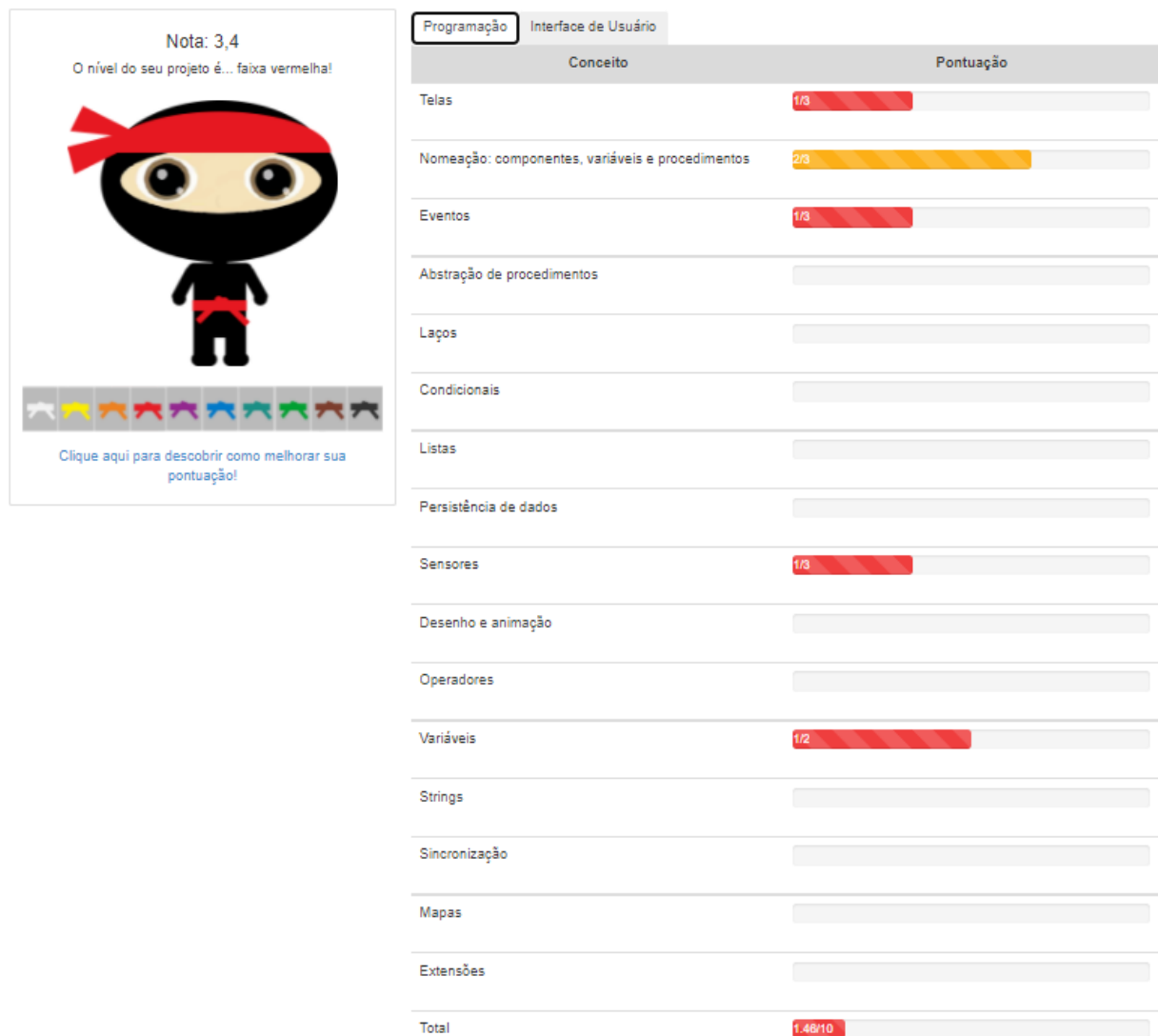


Figura 21 - Resultado de programação de projeto App Inventor no CodeMaster.

Nota: 3,4

O nível do seu projeto é... faixa vermelha!




Clique aqui para descobrir como melhorar sua pontuação!

Programação	Interface de Usuário
Layout	10.0/10
Tamanho de componentes alvos de toque	2/2
Formato dos botões	2/2
Tamanho consistente de botões	2/2
Densidade da tela	2/2
Tipografia	10.0/10
Família da fonte	2/2
Tamanho da fonte de botões	2/2
Tamanho da fonte de componentes	2/2
Uso de itálico	2/2
Escrita	7.5/10
Capitalização de botões	
Capitalização de sentenças	2/2
Texto padrão dos componentes	2/2
Evitar dois pontos	2/2
Evitar ponto final	2/2
Comprimento do texto de botões	1/2
Cores	2.5/10

Figura 22 - Resultado de Interface Gráfica de projeto App Inventor no CodeMaster.

5.2.2 Teste de funcionalidade do menu Aluno - Snap!

Foram agrupados os seguintes requisitos:

RF02 - Realizar *upload* de um projeto Snap!

RF07 - Analisar Projeto Snap!

RF09 - Avaliar Projeto Snap!

RF11 - Apresentar avaliação detalhada do Projeto Snap!.

No menu Aluno, o usuário deverá fazer o upload do arquivo Snap!, selecionando o arquivo.xml que deseja avaliar, e clicar no botão avaliar, conforme Figura 24.



Figura 23 - Upload de projeto Snap! no CodeMaster.

O usuário será redirecionado para tela de resultado, que apresentará as pontuações, nota final e faixa, conforme Figura 25.



Figura 24 - Resultado de projeto Snap! no CodeMaster.

5.2.3 Teste de funcionalidade do menu Professor - App Inventor

Foram agrupados os seguintes requisitos:

RF04 - Realizar *Upload* de um conjunto de projetos App Inventor

RF12 - Apresentar Avaliação dos projetos App Inventor de forma resumida.

Para realizar o *upload* de um conjunto de projetos App Inventor, é necessário criar uma turma e selecionar o tipo de projeto “App Inventor” conforme Figura 26.

The screenshot shows the CodeMaster website interface. At the top left is the CodeMaster logo, and at the top right is a 'Logout' link. A green navigation bar contains the following menu items: Home, Aluno, Professor, Administrador, and Machine Learning para Todos. The main content area is titled 'Minhas Turmas' and features a progress indicator with three steps: 1 (Turma), 2 (App Inventor/Snap!), and 3 (Arquivos). Below the progress indicator, there is a form for creating a class. The 'Nome da Turma:' field contains the text 'Teste Funcional'. Under 'Linguagem:', the 'App Inventor' radio button is selected, and the 'Snap!' radio button is unselected. The footer of the page includes logos for GQS, INCoD, ine, and UFSC, along with the text 'COMPUTAÇÃO NA ESCOLA' and links for 'Sobre', 'Política de Privacidade', and 'Termos de Serviço'.

Figura 25 - Criação de turmas App Inventor do CodeMaster.

Na aba seguinte serão apresentados os conceitos que serão avaliados, referentes a Programação e Interface de Usuário, o usuário pode manter todos selecionados, ou escolher apenas o que deseja avaliar, conforme Figura 27, e ir para aba de *upload* do arquivo.

Minhas Turmas

1 Turma 2 App Inventor/Snap! 3 Arquivos

Programação Interface de Usuário

Programação

Conceitos avaliados em projetos App Inventor

- Telas
- Eventos
- Nomeação de Componentes
- Abstração de Procedimentos
- Laços
- Condicionais
- Listas
- Persistência de Dados
- Sensores
- Desenho e Animação
- Operadores
- Variáveis
- Strings
- Sincronização
- Mapas
- Extensões

Figura 26 - Seleção de conceitos para avaliação do App Inventor do CodeMaster.

Na aba Arquivos, o usuário deverá fazer o *upload* dos arquivos App Inventor, selecionando os arquivos.aia que deseja avaliar, e clicar no botão avaliar, conforme Figura 28.

Minhas Turmas

- 1 Turma
- 2 App Inventor/Snap!
- 3 Arquivos

Avalie seu projeto

Primeiro clique no botão "Escolher arquivos" e selecione todos os projetos

2 arquivos

Então clique no botão "Avaliar"

Figura 27 - Upload dos arquivos App Inventor da turma do CodeMaster.

O usuário será redirecionado para tela de resultado, que apresentará as pontuações, nota final e faixa, conforme Figura 29.

Avaliações de projetos App Inventor

Total	Programação	Interface de Usuário		
Projeto	Nota em Programação	Nota em Interface de Usuário	Nota Final	Nível
HelloPurr.aia	7.27	0.49	2.5239999999999996	faixa laranja
MathExperienciaaaaa.aia	3	2.44	2.6079999999999997	faixa laranja



Figura 28 - Resultado da turma App Inventor do CodeMaster.

5.2.4 Teste de funcionalidade do menu Professor - Snap!

Foram agrupados os seguintes requisitos:

RF05 - Realizar *upload* de conjunto de projetos Snap!

RF13 - Apresentar avaliação resumida dos projetos Snap!.

Para realizar o *upload* de um conjunto de projetos Snap!, é necessário criar uma turma e selecionar o tipo de projeto “Snap!” conforme Figura 30.

Minhas Turmas

1 ————— 2 ————— 3

Turma App Inventor/Snap! Arquivos

Nome da Turma:

Linguagem:

App Inventor

Snap!

Figura 29 - Criação de turmas Snap! do CodeMaster.

Na aba seguinte serão apresentados os conceitos que serão avaliados, o usuário pode manter todos selecionados, ou escolher apenas o que deseja avaliar, conforme Figura 31, e ir para aba de *upload* do arquivo.

Minhas Turmas

1 2 3

Turma App Inventor/Snap! Arquivos

Programação

Conceitos avaliados em projetos Snap!

- Lógica
- Paralelismo
- Interatividade com o usuário
- Representação de dados
- Controle de fluxo
- Sincronização
- Abstração
- Operadores

Figura 30 - Seleção de conceitos para avaliação do Snap! do CodeMaster.

Na aba Arquivos, o usuário deverá fazer o *upload* dos arquivos Snap!, selecionando os arquivos.xml que deseja avaliar, e clicar no botão avaliar, conforme Figura 32.

Minhas Turmas

1 Turma 2 App Inventor/Snap! 3 Arquivos

Avalie seu projeto

Primeiro clique no botão "Escolher arquivos" e selecione todos os projetos

Escolher arquivos 2D rectangle collision.xml

Então clique no botão "Avaliar"

Avaliar

Figura 31 - Upload dos arquivos Snap! da turma do CodeMaster.

O usuário será redirecionado para tela de resultado, que apresentará as pontuações, nota final e faixa, conforme Figura 33.

The screenshot shows the CodeMaster website interface. At the top left is the CodeMaster logo, and at the top right is a 'Logout' link. Below the logo is a green navigation bar with links for 'Home', 'Aluno', 'Professor', 'Administrador', and 'Machine Learning para Todos'. The main content area is titled 'Avaliações de projetos Snap!' and contains a table with the following data:

Projeto	Lógica	Paralelismo	Interatividade com o usuário	Representação de dados	Controle de fluxo	Sincronização	Abstração	Operadores	Pontuação total	Nota	Nível
2D rectangle collision.xml	3	3	1	3	2	0	2	3	17	7.083333333333334	faixa verde

At the bottom of the page, there is a green footer bar containing logos for GQS, INCoD, ine, and UFSC, along with the text 'COMPUTAÇÃO NA ESCOLA' and links for 'Sobre', 'Política de Privacidade', and 'Termos de Serviço'.

Figura 32 - Resultado dos projetos Snap! da turma do CodeMaster.

5.2.5 Teste da funcionalidade cadastrar Professor

RF14 - Realizar o Cadastro de Professor.

Para cadastrar um novo professor, é necessário clicar no menu Professor, na tela de login, clicar em “Ainda não sou cadastrado”, conforme Figura 34, para ser redirecionado para tela de cadastro.

Home Aluno **Professor** Administrador Machine Learning para Todos

Acesso para professores cadastrados

toni@codemaster.ufsc.br

.....

Login

[Esqueci minha Senha](#)
[Ainda não sou cadastrado](#)

QOS INCoD ine UFSC

COMPUTAÇÃO NA ESCOLA

[Sobre](#) [Política de Privacidade](#) [Termos de Serviço](#)

Figura 33 - Tela inicial do menu Professor do CodeMaster.

Após ser redirecionado, basta preencher as informações solicitadas e aceitar os termos de uso, conforme Figura 35, para realizar o cadastro do Professor.

Cadastro para professores

Toni

toni@codemaster.ufsc.br

UFSC

TCC

Brasil

SC

Florianópolis

.....

.....

Li e aceito os termos de serviço.

Cadastrar

Figura 34 - Tela de cadastro de Professor do CodeMaster.

5.2.6 Teste da funcionalidade login Professor

RF15 - Realizar Login de Professor.

Para realizar o login, basta clicar no Menu “Professor”. Se o usuário não estiver logado, será solicitado login e senha. Caso o usuário esteja logado, apresentará a

possibilidade de criar nova turma ou visualizar as turmas já cadastradas, conforme imagens 36 e 37:

The image shows the CodeMaster website interface. At the top left is the CodeMaster logo. At the top right is a 'Logout' link. Below these is a green navigation bar with the following menu items: Home, Aluno, Professor, Administrador, and Machine Learning para Todos. The main content area features a login form titled 'Acesso para professores cadastrados'. The form contains two input fields: the first is for an email address, with 'toni@codemaster.ufsc.br' entered; the second is for a password, shown as '*****'. Below the password field is a green 'Login' button. Underneath the button are two links: 'Esqueci minha Senha' and 'Ainda não sou cadastrado'. At the bottom of the page is a green footer bar containing logos for GQS, INCoD, ine, and UFSC on the left, and the 'COMPUTAÇÃO NA ESCOLA' logo on the right. Below the logos are the links 'Sobre', 'Política de Privacidade', and 'Termos de Serviço'.

Figura 35 - Tela de login de Professor do CodeMaster.

Minhas Turmas

1 ————— 2 ————— 3

Turma App Inventor/Snap! Arquivos

Nome da Turma:

Linguagem:

App Inventor

Snap!

Figura 36 - Tela do Professor logado no CodeMaster.

5.2.7 Teste da funcionalidade login de Administrador

RF16 - Realizar login de Administrador.

Ao realizar o *login* de Administrador, usuário é redirecionado para tela de Gerenciamento de consulta de projetos, conforme Figuras 37 e 38:

Acesso para administradores cadastrados

Login

[Esqueci minha Senha](#)

Figura 37 - Tela após o login do Administrador no CodeMaster.

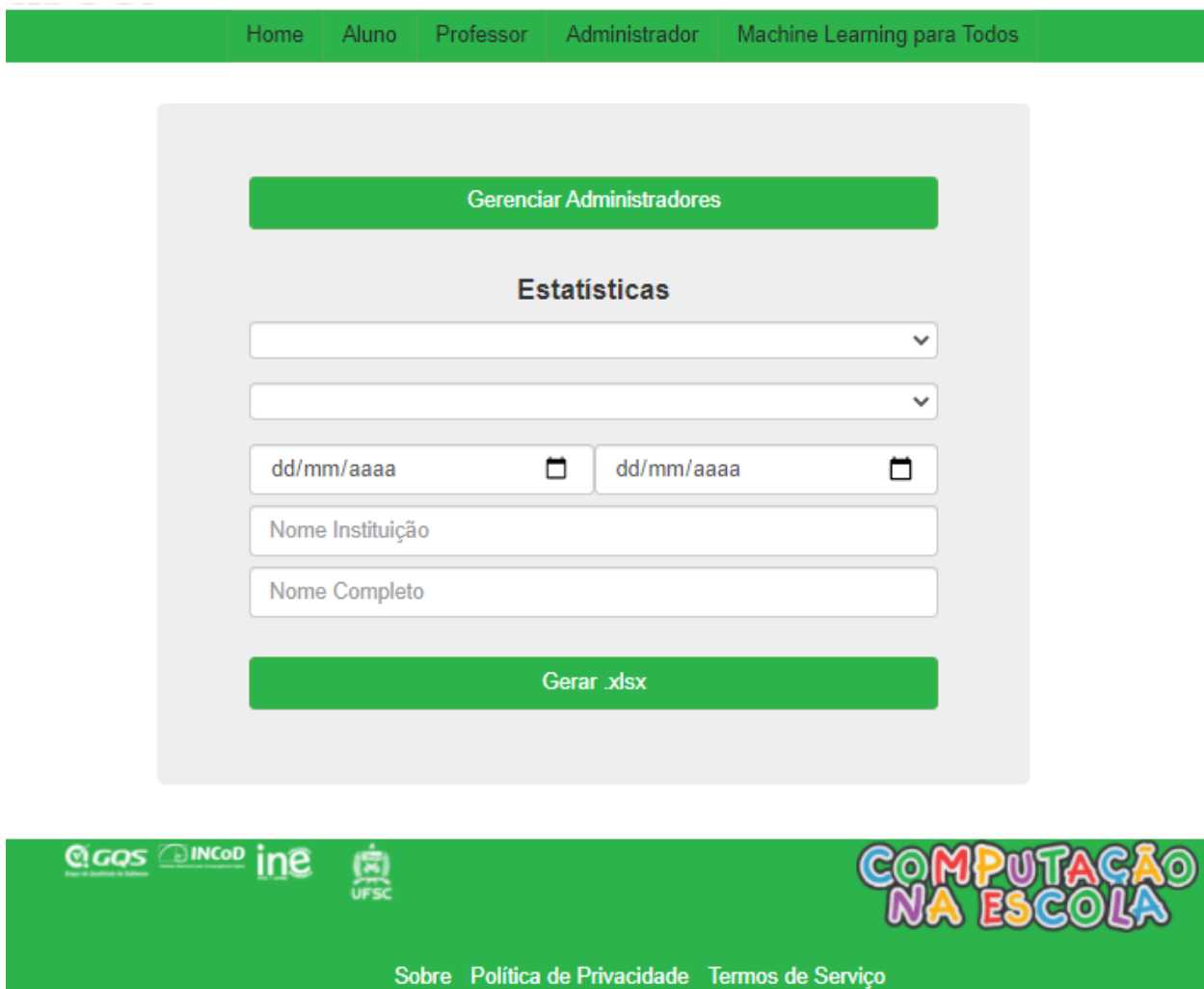


Figura 38 - Tela do Administrador logado

5.2.9 Teste da funcionalidade logout Professor

RF17 - Realizar Logout de Professor.

Ao realizar o Logout de Professor, usuário é redirecionado para tela de login do Professor, conforme Figura 39:

Home Aluno Professor Administrador Machine Learning para Todos

Acesso para professores cadastrados

toni@codemaster.ufsc.br

Login

[Esqueci minha Senha](#)

[Ainda não sou cadastrado](#)

GOS INCoD ine UFSC

COMPUTAÇÃO NA ESCOLA

[Sobre](#) [Política de Privacidade](#) [Termos de Serviço](#)

Figura 39 - Tela após o logout do Professor no CodeMaster.

5.2.10 Teste da funcionalidade logout de Administrador

RF18 - Realizar *logout* de Administrador.

Ao realizar o Logout de Administrador, usuário é redirecionado para tela de login do Administrador, conforme Figura 38, que representa o *logout* do Professor.

5.2.11 Teste da funcionalidade manter registros da avaliação anônima

RF20 - Manter Registro de Avaliação de Usuário Anônimo.

Registros continuam sendo persistidos na tabela Projeto da Base de dados.

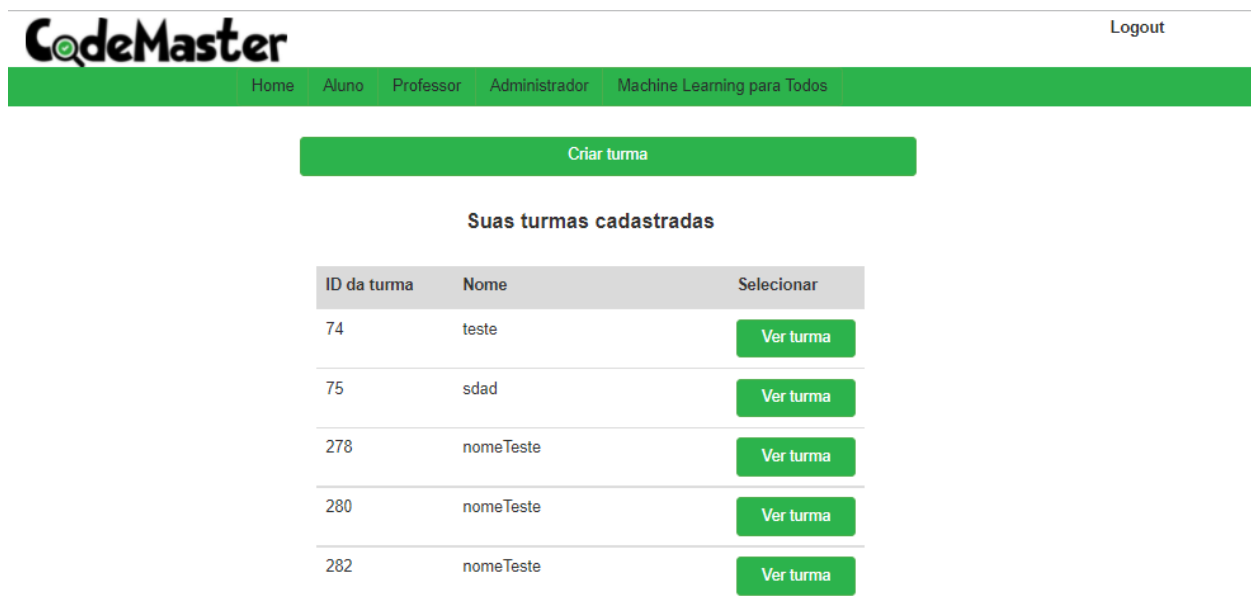
5.2.12 Teste da funcionalidade manter registros das avaliação do Professor

Foram agrupados os seguintes requisitos:

RF19 - Manter Registro de Avaliações do Professor,

RF21 - Apresentar todas as Avaliações do Professor.

Para visualizar todas as avaliações do professor, o usuário precisa estar logado e clicar no botão “Minhas Turmas” da tela inicial do menu Professor e será direcionado para listagem de turmas do professor conforme Figura 39.



The screenshot shows the CodeMaster web application interface. At the top left is the CodeMaster logo, and at the top right is a 'Logout' link. Below the navigation bar is a green button labeled 'Criar turma'. The main content area displays a table titled 'Suas turmas cadastradas' with the following data:

ID da turma	Nome	Selecionar
74	teste	Ver turma
75	sdad	Ver turma
278	nomeTeste	Ver turma
280	nomeTeste	Ver turma
282	nomeTeste	Ver turma

Figura 40 - Tela do Professor logado no CodeMaster.

Ao escolher a turma e selecionar o botão “Ver Turma”, usuário é direcionado para a tela de resultado da Turma, conforme Figura 40:

Avaliações de projetos App Inventor

Total	Programação	Interface de Usuário		
Projeto	Nota em Programação	Nota em Interface de Usuário	Nota Final	Nível
HelloPurr.aia	7.27	0.49	2.5239999999999996	faixa laranja
MathExperianceaaaa.aia	3	2.44	2.6079999999999997	faixa laranja

Figura 41 - Apresentação dos resultados das turmas no CodeMaster.

5.2.13 Teste da funcionalidade apresentar estatísticas de avaliações

RF22 - Apresentar Estatísticas de Avaliações.

Este requisito está em desenvolvimento quando da entrega deste documento e será finalizado até a entrega da versão final do TCC.

6 CONCLUSÃO

Neste trabalho é apresentada a refatoração do sistema *CodeMaster*, tendo como principal objetivo a atualização das tecnologias utilizadas na implementação do sistema e alteração da estrutura do projeto, separando completamente as camadas de *back-end* e o *front-end*, para melhorar sua manutenibilidade e torná-lo mais atrativo para o desenvolvimento de novas funcionalidades.

Inicialmente, foram realizadas análises dos principais conceitos relacionados a este trabalho, como o estudo da própria aplicação *CodeMaster*, conceitos de avaliação de código no ensino de computação e como o *CodeMaster* avalia um projeto e quais tecnologias foram utilizadas na aplicação. Estudos sobre conceitos como manutenção de *software*, manutenção perfectiva e refatoração também foram realizados no desenvolvimento do projeto.

Em seguida foi realizado um estudo da arquitetura atual do sistema e análise das principais necessidades de manutenção, como a separação completa do *back-end* e *front-end* e a modernização das tecnologias, seguindo alguns requisitos como, manter o *back-end* em *Java* e utilizar tecnologias atuais com comunidades ativas, e se possível, tecnologias que o autor tenha familiaridade. Após este estudo foi iniciada a refatoração da aplicação, com a configuração do ambiente de desenvolvimento e criação dos projetos que substituirão o módulo de apresentação do sistema legado, sendo um projeto para o *font-end*, desenvolvido em *Angular 12*, e outro projeto *Java*, utilizando *SpringBoot*, responsável pela persistência dos dados, regras de negócio, comunicação com o *front-end* e com o *RESTGrader*, o motor de regras do sistema.

Comparando a arquitetura e as tecnologias utilizadas após a refatoração, que mesmo durante o processo de desenvolvimento, já permitiu que outro projeto fosse desenvolvido em paralelo, adicionando novas funcionalidades ao sistema, percebe-se que a modernização do sistema possibilita que diferentes funcionalidades possam ser desenvolvidas simultaneamente sem grandes problemas devido à alta coesão e ao baixo acoplamento dos módulos.

Com esta refatoração, espera-se que novas funcionalidades sejam incorporadas ao sistema, como sugestão, a avaliação por pares e compartilhamento de tarefas com colegas e a comunidade, com intuito de facilitar a aprendizagem do usuário, permitindo que ele se desafie com novos projetos, este era o objetivo inicial deste trabalho.

Algumas melhorias no *layout* do sistema podem ser realizadas, principalmente na apresentação dos resultados das turmas, na visão dos professores, além de melhorar a apresentação das informações, acrescentar uma aba de gráficos para apresentar de forma mais amigável os resultados gerais da turma.

REFERÊNCIAS

APP INVENTOR, 2021. Disponível em: <<http://appinventor.mit.edu/explore/>>. Acesso em: fevereiro de 2021.

BILAL, M., CHAN, P., MEDDINGS, F., & KONSTADOPOULOU, A. **SCORE: An advanced assessment and feedback framework with a universal marking scheme in higher education**. In Proc. of the Int. Conf. on Education and e-Learning Innovations, Sousse/Tunísia, 2012, 1-6.

BISBAL, J.; LAWLESS, D.; WU, B.; GRIMSON, J. **Legacy Information Systems: Issues and Directions**, 1999 IEEE Softw., IEEE Computer Society Press, Los Alamitos.

BLACK, P., & WILIAN, D. **Assessment and classroom learning**. Assessment in Education: Principles, Policy & Practice, 5(1), 1998.

BRENNAN, K., RESNICK, M. **New frameworks for studying and assessing the development of computational thinking**. Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada, 2012.

CNE. GQS/INCoD/INE/UFSC. Iniciativa Computação na Escola, 2013. Disponível em: . Acesso em: junho de 2019.

COMELLA-DORDA, S.; WALLNAU, K.; SEACORD, R.; ROBERT, J. **A Survey of Legacy System Modernization Approaches**. Pittsburgh, PA, 2000.

CSTA. **ACM. CSTA K –12 Computer Science Standards**, 2017.

DEITEL H. M., DEITEL P.J. **Java Como Programar**. Sexta Edição, Prentice Hall. 2004.

DEMETRIO, M. F. (2017) **Desenvolvimento de um analisador e avaliador de código de App Inventor para ensino de computação**. 2017. Trabalho de Conclusão do Curso de Bacharel em Ciências da Computação da Universidade Federal de Santa Catarina, Florianópolis, Brasil.

DR. SCRATCH. **Dr.Scratch** – Explorar. Disponível em: <<https://scratch.mit.edu/>>. Acesso em: Junho 2019.

ESERYEL, D., IFENTHALER, D., XUN, G. **Validation study of a method for assessing complex ill-structured problem solving by using causal representations**. Educational Technology Research and Development, 61, 443–463, 2013.

FREITAS, C. Bruno **Modernização de Sistemas Legados para Disponibilização em Dispositivos Móveis com Arquitetura Baseada em microservices**. 2017. Dissertação apresentada à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco - Cin/UFPE.

ISO/IEC/IEEE 24765:2017 **Systems and software engineering-Vocabulary**

JERSEY, **RESTful Web Services in Java**. 2017. Disponível em: < <https://eclipse-ee4j.github.io/jersey/>>. Acesso em: Março de 2022.

JUSTEN, Karla A. **Desenvolvimento de um Analisador de Design de Interface no Contexto do Ensino de Computação com o App Inventor**. 2019. Trabalho de

Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade Federal de Santa Catarina.

KANGASALO, M. **Legacy application modernization with REST wrapping**. 2016. Tese (Doutorado) — University of Helsinki.

LACOB, C., FAILY, S., **Redesigning an undergraduate software engineering course for a large cohort**, presented at the Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training, Gothenburg, Sweden, 2018.

MARTINS, P. H. R. Osvaldo. **Desenvolvimento de um Modelo para Avaliação da Estética Visual de Interfaces de Usuários de Aplicativos Usando Deep Learning**. 2019. Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade Federal de Santa Catarina.

MORENO-LEÓN, J.; ROBLES, G.; **Computer programming as an educational tool in the English classroom a preliminary study**. In: Proceedings of 2015 IEEE Global Engineering Education Conference, Tallinn, Estonia, 2015.

MOZILLA, **Sobre JavaScript**. 2022. Disponível em: < https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript>. Acesso em: Março de 2022.

NILSSON, S. **Application modernization: approaches, problems and evaluation**. 2015. Tese (Doutorado) — Umea University.

PELLE, Rafael (2018) **Desenvolvimento de um analisador de código para snap! voltado ao ensino de computação na educação básica**. 2018. Trabalho de Conclusão do Curso de Bacharel em Ciências da Computação da Universidade Federal de Santa Catarina, Florianópolis, Brasil.

PRESSMAN, Roger S. **Software engineering: a practitioner's approach**. Palgrave macmillan, 2005.

RASHKOVITS, R; LAVY, I. **FACT: A Formative Assessment Criteria Tool for the Assessment of Students' Programming Tasks**. In: Proceedings of the World Congress on Engineering. Londres, UK, 2013.

SHERMAN, M; MARTIN, F. **The assessment of mobile computational thinking**. Journal of Computing Sciences in Colleges, v. 30, n. 6, 2015.

SNAP, 2021. Disponível em: <<https://snap.berkeley.edu/explore>>. Acesso em: fevereiro de 2021.

SOMMERVILLE, Ian. **Software engineering 9th Edition**. PEARSON, 2011.

VISAGGIO, G. **Ageing of a data-intensive legacy system: symptoms and remedies**. **Journal of Software Maintenance and Evolution: Research and Practice**, 2001. Wiley Online Library.

WEIDERMAN, N.; NORTHROP, L.; SMITH, D.; TILLEY, S.; WALLNAU, K. **Implications of Distributed Object Technology for Reengineering**, 1997.

WHITTAKER, C. R., SALEND, S. J., DUHANEY, D. **Creating instructional rubrics for inclusive classrooms. Teaching Exceptional Children**, 34(2), 8-13, 2001.

ZEN, K., Iskandar, D.N.F.A., Linang, O. (2011). **Using Latent Semantic Analysis for automated grading programming assignments.** Proceedings of the 2011 International Conference on Semantic Technology and Information Retrieval, Putrajaya, Malaysia, pages 82 –88.

APÊNDICE A – Código-Fonte

<https://codigos.ufsc.br/100000000394729/CodeMaster>.

APÊNDICE B - Artigo

Refatoração da Ferramenta CodeMaster

Toni Marcos Schmitt

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

toni.schmitt@grad.ufsc.br

***Abstract.** With the introduction and advancement of software in all areas of society, the demand for trained professionals for its development also grows. Then there is the need to insert the teaching of computing in basic education, so that young people feel closer and more interested in software development. To facilitate the insertion of lay users into the universe of software development, there are programming tools based on blocks, such as App Inventor, which enable the development of apps for mobile devices for Android. In this context, the CodeMaster tool aims to support the evaluation of projects developed by students. The original version of CodeMaster analyzes and evaluates projects developed with App Inventor and Snap!, in addition to images. Thus, the objective of this work is to carry out the code refactoring of CodeMaster, updating some outdated technologies that are used in its source code, such as Servlets and JSP, and separating the back-end and front-end layers. With this perfective maintenance, using current frameworks for the system, it is expected to facilitate code maintainability and the development of new features in the future.*

Resumo. Com a introdução e avanço de softwares em todas as áreas da sociedade, a demanda por profissionais capacitados para seu desenvolvimento também cresce. Surge então a necessidade de inserir o ensino de computação na educação básica, para que as jovens se sintam mais próximas e interessadas no desenvolvimento de softwares. Para facilitar a inserção de usuários leigos ao universo do desenvolvimento de softwares, existem ferramentas de programação baseadas em blocos, como o App Inventor, que possibilitam o desenvolvimento de apps para dispositivos móveis para Android. Neste contexto, a ferramenta CodeMaster visa apoiar na avaliação de projetos desenvolvidos pelos alunos. A versão original do CodeMaster analisa e avalia projetos desenvolvidos com o App Inventor e Snap!, além de imagens. Deste modo, o objetivo deste trabalho é realizar a refatoração de código do CodeMaster, atualizando algumas tecnologias defasadas que são utilizadas em seu código fonte, como Servlets e JSP e, separando as camadas de back-end e front-end. Com esta manutenção perfectiva, utilizando frameworks atuais para o sistema, espera-se facilitar a manutenibilidade do código e o desenvolvimento de novas funcionalidades no futuro.

1. Introdução

A inclusão do ensino de computação nas escolas esbarra, dentre outros fatores, na falta de ferramentas de suporte aos professores na avaliação de trabalhos práticos de programação dos estudantes (MORENO-LÉON et al., 2015). Esta situação é agravada pela falta de formação dos professores do Ensino Básico em áreas relacionadas a tecnologia da informação e comunicação. Esta lacuna gera um grande esforço para colocar em prática o ensino de computação na escola, seja na definição das atividades e até mesmo na avaliação das atividades práticas desenvolvidas pelos alunos (ESERYEL et al., 2013).

Porém este cenário está se modificando. Atualmente há iniciativas que entendem que todos os alunos em todas as escolas devem ter a oportunidade de aprender computação e visam aumentar o ensino de computação no Ensino Fundamental e Médio (CNE, 2019). Uma das formas de ensinar programação é por meio de trabalhos práticos, em que os alunos desenvolvem programas utilizando linguagens de programação. Neste contexto, pode-se utilizar as ferramentas App Inventor (APP INVENTOR, 2022) e Snap! (SNAP, 2022) entre outros, que são ambientes para programação visual baseada em blocos que permite que pessoas, sem conhecimento computacional, comecem a programar e construir aplicativos para dispositivos Android ou ferramentas similares que possuem o mesmo objetivo.

No contexto de falta de formação específica dos professores, a ferramenta CodeMaster (DEMETRIO, 2017) auxilia na avaliação dos trabalhos desenvolvidos pelos alunos, permitindo a avaliação automatizada de vários conceitos aumentando a objetividade e eliminando qualquer favoritismo ou inconsistência (ZEN et al., 2011).

Entretanto, a ferramenta CodeMaster foi inicialmente criada em 2016 e, desde então, diversas funcionalidades adicionais foram sendo implementadas por outros trabalhos de pesquisa, tais como (Pelle, 2018; Justen, 2019; Martins, 2019). Assim, com a incorporação de diversas funcionalidades, o código-fonte sofreu degradação, perdendo sua manutenibilidade. Manutenibilidade é a facilidade com que um sistema ou componente de *software* pode ser modificado para alterar ou adicionar recursos, corrigir falhas ou defeitos, melhorar o desempenho ou outros atributos ou adaptar-se a um ambiente alterado (ISO/IEC/IEEE 24765, 2017).

2. Objetivo

O objetivo principal deste projeto é realizar a manutenção perfectiva da ferramenta web de análise e avaliação automatizada CodeMaster, melhorando a manutenibilidade e atualizando as tecnologias utilizadas no código-fonte.

3. Fundamentação Teórica

CodeMaster é uma ferramenta web para análise e avaliação de código baseado em linguagens de programação em blocos para dar suporte ao ensino de computação, inicialmente voltada para

aplicativos desenvolvidos com App Inventor (DEMETRIO, 2017) e, posteriormente foi aprimorado para também analisar e avaliar aplicativos desenvolvidos com Snap! (PELLE, 2018). A análise de código aplicada no CodeMaster é focada no grau de atendimento de objetivos de aprendizagem. O CodeMaster exige flexibilidade para que o analisador faça a análise livre do código, pois podem haver diversas soluções corretas para um mesmo problema (DEMETRIO, 2017).

A análise e avaliação do programa baseado em atributos mensuráveis que podem ser extraídos do programa, avaliando assim o aprendizado esperado dos alunos. Esta métrica, chamada de rubrica, utiliza medidas descritivas para separar os níveis de desempenho em uma determinada tarefa, delineando os vários critérios associados às atividades de aprendizado (WHITTAKER et al., 2001), e permite o feedback instrucional, definindo a pontuação de cada critério.

Refatorações são transformações parametrizadas do código-fonte de um sistema destinadas a melhorar a estrutura de um sistema em relação a objetivos expressos informalmente, como manutenibilidade, alterabilidade, legibilidade, desempenho ou demandas de memória. A refatoração geralmente preserva o comportamento do sistema (ARORA et al., 2011).

Segundo Fowler (1999) refatoração é o “processo de alterar um sistema de *software* de tal forma que não altere o comportamento externo do código, mas melhore sua estrutura interna”. Consiste em uma forma disciplinada de “limpar” o código, melhorando o design do código depois que ele foi escrito e minimizando as chances de introdução de bugs.

4. Refatoração do CodeMaster

A arquitetura e algumas tecnologias empregadas no CodeMaster V1.0 são consideradas defasadas, como *Servlets* e *JSP*, gerando grandes dificuldades para a evolução da ferramenta devido à falta de desenvolvedores aptos e com conhecimento nestas tecnologias.

A completa separação das camadas *front-end* e *back-end* melhoram a manutenibilidade do código, sendo que novos desenvolvedores não necessariamente precisam ter conhecimento em ambas as camadas. Com a refatoração, foram utilizadas tecnologias atuais e com grande aceitação no mercado, como o *SpringBoot* e *Angular*, o que pode ser considerado como um atrativo para que novos desenvolvedores possam propor novas funcionalidades para evoluir a aplicação.

A separação do *software* em camadas de *front-end* e *back-end* é uma estratégia muito utilizada no desenvolvimento de *software* por vários motivos, como simplificar o desenvolvimento e a manutenção da aplicação e a possibilidade de serem desenvolvidos simultaneamente por diferentes equipes. O fato de possuir códigos separados e independentes de tecnologia permite a alteração de uma das tecnologias sem que haja a necessidade de alterar a outra camada, além da possibilidade de hospedar o *front-end* em um servidor e o *back-end* em outro servidor.

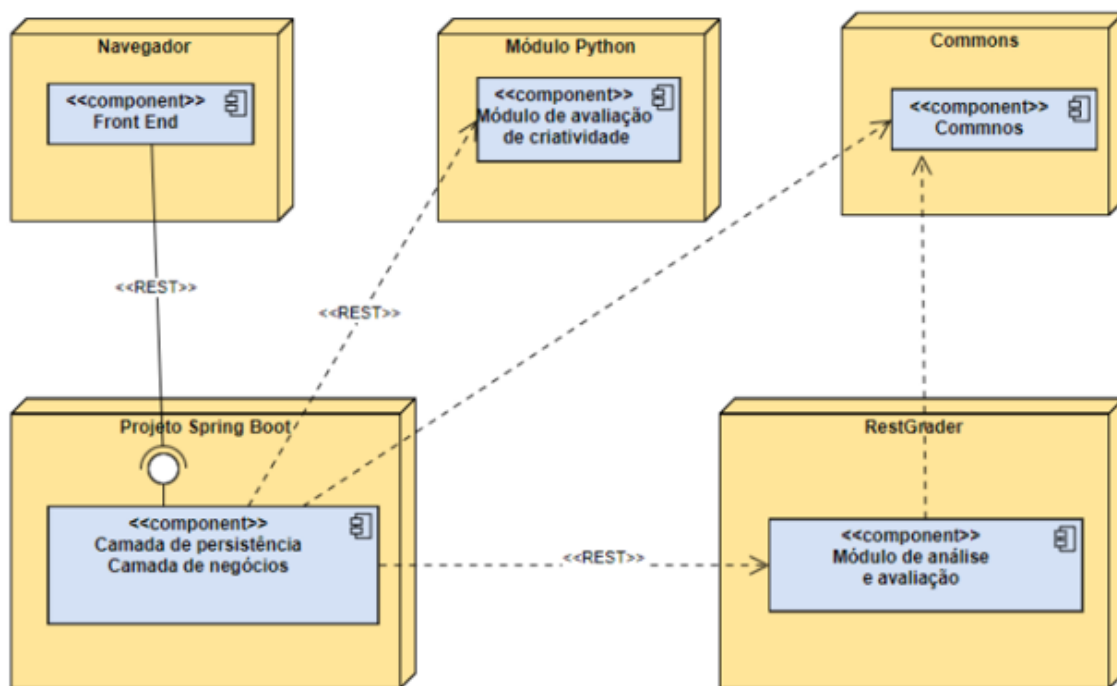


Figura 1 - Arquitetura do CodeMaster

O módulo de análise e avaliação, denominado *RESTGrader* e o *Commons* não tiveram alterações significativas, pequenas correções para padronização. O módulo de apresentação do *CodeMaster* V1.0 foi dividido em dois projetos distintos, separando *front-end* de *back-end*. O *front-end* é responsável pela interface com o usuário, recebendo parâmetros e apresentando os resultados das avaliações que são realizadas no *back-end*. O *back-end* é responsável pela camada de negócios, persistência de dados e interação com os módulos de análise.

5. Conclusão

Foi realizado a refatoração do sistema *CodeMaster*, tendo como principal objetivo a atualização das tecnologias utilizadas na implementação do sistema e alteração da estrutura do projeto, separando completamente as camadas de *back-end* e o *front-end*, para melhorar sua manutenibilidade e torná-lo mais atrativo para o desenvolvimento de novas funcionalidades.

6. Referências

APP INVENTOR, 2021. Disponível em: <<http://appinventor.mit.edu/explore/>>. Acesso em: fevereiro de 2021.

CNE. GQS/INCoD/INE/UFSC. Iniciativa Computação na Escola, 2013. Disponível em: Acesso em: junho de 2019.

DEMETRIO, M. F. (2017) **Desenvolvimento de um analisador e avaliador de código de App Inventor para ensino de computação**. 2017. Trabalho de Conclusão do Curso de Bacharel em Ciências da Computação da Universidade Federal de Santa Catarina, Florianópolis, Brasil.

ESERYEL, D., IFENTHALER, D., XUN, G. **Validation study of a method for assessing complex ill-structured problem solving by using causal representations**. Educational Technology Research and Development, 61, 443–463, 2013.

JUSTEN, Karla A. **Desenvolvimento de um Analisador de Design de Interface no Contexto do Ensino de Computação com o App Inventor**. 2019. Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade Federal de Santa Catarina.

MARTINS, P. H. R. Osvaldo. **Desenvolvimento de um Modelo para Avaliação da Estética Visual de Interfaces de Usuários de Aplicativos Usando Deep Learning**. 2019. Trabalho de Conclusão de Curso. (Graduação em Ciência da Computação) – Universidade Federal de Santa Catarina.

MORENO-LEÓN, J.; ROBLES, G.; **Computer programming as an educational tool in the English classroom a preliminary study**. In: Proceedings of 2015 IEEE Global Engineering Education Conference, Tallinn, Estonia, 2015.

PELLE, Rafael (2018) **Desenvolvimento de um analisador de código para snap! voltado ao ensino de computação na educação básica**. 2018. Trabalho de Conclusão do Curso de Bacharel em Ciências da Computação da Universidade Federal de Santa Catarina, Florianópolis, Brasil.

SNAP, 2021. Disponível em: <<https://snap.berkeley.edu/explore>>. Acesso em: fevereiro de 2021.

ZEN, K., Iskandar, D.N.F.A., Linang, O. (2011). **Using Latent Semantic Analysis for automated grading programming assignments**. Proceedings of the 2011 International Conference on Semantic Technology and Information Retrieval, Putrajaya, Malaysia, pages 82 –88.