UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ESTEVAN FREDERICO PASQUETTA JANTSK
RICARDO TRIZZOLINI PIEKARSKI

DESENVOLVIMENTO DE UM SISTEMA DE INFORMAÇÃO PARA AVALIAÇÃO DE ALUNOS DA EDUCAÇÃO FUNDAMENTAL UTILIZANDO DESENVOLVIMENTO BASEADO EM COMPORTAMENTO

TRABALHO DE CONCLUSÃO DE CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

CURITIBA
2014

ESTEVAN FREDERICO PASQUETTA JANTSK
RICARDO TRIZZOLINI PIEKARSKI

DESENVOLVIMENTO DE UM SISTEMA DE INFORMAÇÃO PARA AVALIAÇÃO DE ALUNOS DA EDUCAÇÃO FUNDAMENTAL UTILIZANDO DESENVOLVIMENTO BASEADO EM COMPORTAMENTO

Trabalho de conclusão de curso de Bacharelado em Sistemas de Informação da Universidade Tecnológica Federal do Paraná.

Orientador:
Prof. Dr. Paulo Cézar Stadzisz

CURITIBA
2014

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API	Application Programming Interface
BDD	Behavior Drive Development
CSS	Cascading Style Sheets
DCN	Diretrizes Curriculares Nacionais
DSL	Domain Specific Language
EPA	Equipe Pedagógica e Administrativa
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
LDB	Leis de Diretrizes e Bases
MEC	Ministério da Educação
MVC	Model-View-Controller
NASA	National Aeronautics and Space Administration
PNE	Plano Nacional de Educação
RoR	Ruby on Rails
SCM	Source Code Management
SGBD	Sistema Gerenciador de Banco de Dados
SME	Secretaria Municipal da Educação
TCC	Trabalho de Conclusão de Curso
TDD	Test Driven Development
URL	Uniform Resource Locator
UTFPR	Universidade Tecnológica Federal do Paraná
VCS	Version Control System
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

1.1 APRESENTAÇÃO

Este Trabalho de Conclusão de Curso (TCC) aborda o desenvolvimento de uma aplicação web utilizando uma técnica de desenvolvimento baseado em comportamentos (BDD – *Behaviour Driven Develpment*). Este sistema servirá como auxílio à avaliação de desempenho acadêmico de alunos do ensino fundamental em Curitiba. A ideia surgiu da demora e ineficiência das avaliações feitas pelos professores (avaliação somativa) e da análise (avaliação formativa) dessas avaliações pela Equipe Pedagógica e Administrativa (EPA) - composta pela diretora, a vice-diretora e a coordenadora pedagógica das Escolas Municipais de Curitiba. O objeto de estudo deste trabalho é o sistema computacional a ser utilizado pelos professores e pela EPA para fins de avaliar o desempenho acadêmico dos alunos na escola com relação aos conteúdos de cada disciplina e de acordo com os critérios estabelecidos pelo Ministério da Educação (MEC) (BRASIL, 1997) que são utilizados pela Secretaria Municipal da Educação (SME) de Curitiba.

O desenvolvimento baseado em comportamento será empregado utilizando uma ferramenta de mercado que é capaz de compilar as funcionalidades do sistema definidas na especificação de requisitos do sistema em forma de estórias de usuário. Pode-se, assim, diagnosticar se existem os objetos e métodos necessários para que o comportamento de cada funcionalidade seja verdadeiro de acordo com a situação do código no decorrer do desenvolvimento.

1.2 JUSTIFICATIVA

O sistema computacional proposto servirá para que os colaboradores das escolas agilizem o processo de avaliação mensal. Este processo, anteriormente, era extremamente demorado pelo fato de que cada professor necessitava imprimir (ou completar em um computador pessoal diretamente pelo arquivo no *Microsoft Word*) uma folha de avaliação mensal para cada disciplina para cada turma, e então, completá-la com a nota de cada aluno para cada conteúdo passado no período especificado. Além disso, o processo anterior necessitava o armazenamento de cada avaliação em papel requerendo muito espaço físico para o armazenamento das

informações, fazendo com que a utilização dessas informações após armazenadas tornem-se moroso dada a quantidade enorme de folhas de papel em escaninhos presente a cada mês.

Como o sistema computacional proposto utilizará uma arquitetura em nuvem, os colaboradores poderão acessar os dados de qualquer computador que esteja ligado à internet em qualquer momento, sem precisar estar na escola para poder realizar ou analisar as avaliações.

O BDD foi escolhido como método de desenvolvimento de software, pois embora possa trazer grandes benefícios ao processo de desenvolvimento de software, é um assunto ainda pouco explorado no currículo do curso de Bacharelado em Sistemas de Informação da Universidade Tecnológica Federal do Paraná (UTFPR) de Curitiba. Pelo fato de que cada linha do código é programada se, e somente se, houver uma funcionalidade respectiva baseada nas estórias de usuário presente nos requisitos do sistema, o BDD facilita a limitação do escopo do sistema aos requisitos especificados, não deixando margem para código desnecessário.

1.3 OBJETIVO GERAL

O objetivo geral da proposta é desenvolver um sistema computacional utilizando a técnica de desenvolvimento baseado em comportamento a fim de proporcionar uma melhora nas ferramentas de avaliação de desempenho dos alunos feita pelos professores, fazendo com que a realização das avaliações seja feita com mais agilidade e maior eficiência.

1.4 OBJETIVOS ESPECÍFICOS

Mais precisamente, este projeto tem os seguintes objetivos específicos:

- Adquirir conhecimento teórico e prático na utilização de uma abordagem recente de desenvolvimento de software baseada em comportamento (BDD);
- Aprofundar os conhecimentos práticos no uso da linguagem *Ruby* e do *framework Ruby on Rails* em um desenvolvimento de software real;
- Analisar e determinar os comportamentos do sistema para atender as necessidades de um software de avaliação de desempenho de estudantes do ensino fundamental de Curitiba;
- Construir uma versão de avaliação de um sistema de informação para avaliação de desempenho de estudantes do Ensino Fundamental em Curitiba;

1.5 ESTRUTURA DO TRABALHO

Após a introdução do trabalho, o capítulo 2 apresenta o referencial teórico sobre os assuntos envolvidos no projeto, ou seja, desenvolvimento de *software* baseado em testes – ou *test driven development* (TDD), desenvolvimento baseado em comportamento – ou *behaviour driven development* (BDD), *Ruby, Ruby on Rails (RoR),* além de uma breve descrição sobre os conceitos de avaliação de alunos, sistema de ensino por ciclos e o ensino fundamental de nove anos. O capítulo 3 é dedicado à metodologia utilizada no desenvolvimento do sistema. A seguir, no capítulo 4, há uma descrição dos recursos de *hardware* e *software* para a utilização do sistema. O capítulo 5 descreve o contexto atual das escolas em Curitiba. Finalmente, o capítulo 6 cita considerações finais a respeito do trabalho em questão.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico utilizado para o desenvolvimento do Trabalho de Conclusão de Curso. As seções 2.1 a 2.3 apresentam elementos das linguagens adotadas, *framework*, e detalhes da metodologia e técnicas utilizadas. A seção 2.4, e suas subseções, discorre ligeiramente sobre o ensino fundamental em Curitiba e o sistema de avaliação de alunos utilizado em uma escola da rede.

2.1 LINGUAGEM RUBY

Ruby é uma linguagem de programação interpretada, criada em 1994, por Yukihiro Matsumoto e foi inspirada nas linguagens: *Python*, *Perl*, *SmallTalk*, *Lisp* e *Ada* (THOMAS, 2008). A linguagem possui a característica de ser totalmente orientada a objetos, ou seja, até mesmo uma simples *string*, ou um número inteiro irá possuir métodos próprios, a fim de ajudar o programador. Na figura 1 é possível observar o ambiente mais simples possível de desenvolvimento em *Ruby*, utilizando o comando *irb* em um *prompt* de comando e diversas saídas no console após a criação de um objeto no formato *string* e a chamada de alguns de seus métodos nativos.

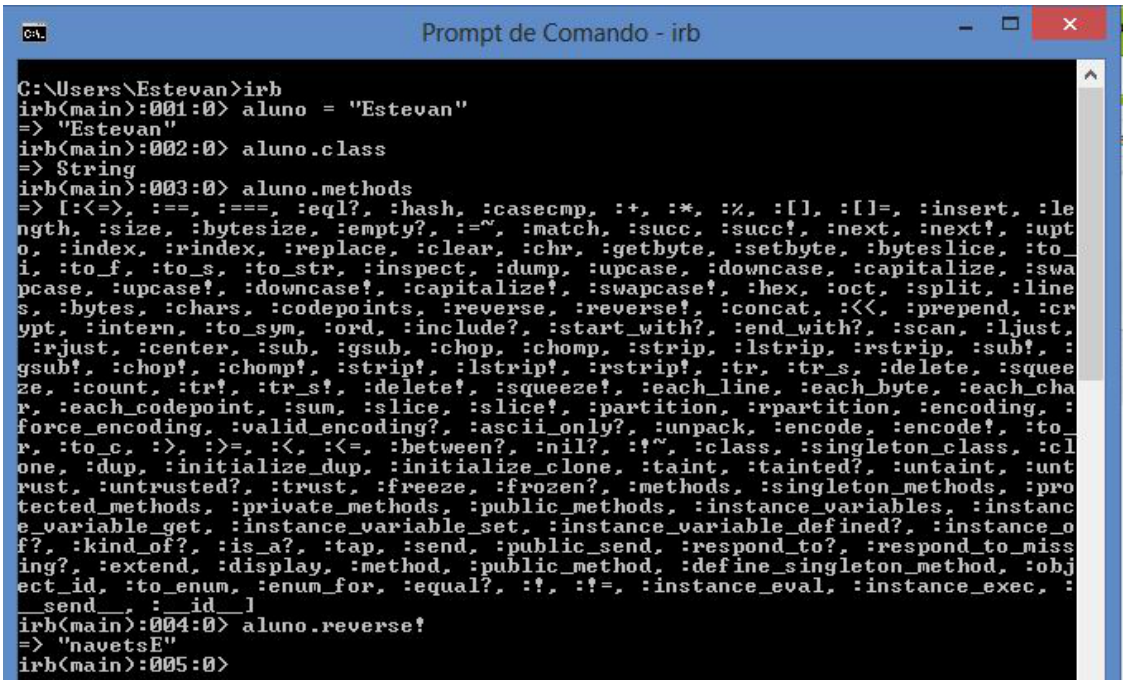


Figura 1 – Criação e manuseio de um objeto em um ambiente de desenvolvimento para Ruby.

No primeiro comando é criada uma variável "aluno", passando-a como valor o nome "Estevan". Como em *Ruby* tudo o que é criado acaba sendo um objeto, por meio do comando "aluno.class", pode-se notar que a variável aluno é uma instância da classe *String*. Em se tratando de uma instância de uma classe, ela possuirá métodos, que podem ser verificados através do comando "aluno.methods**"**. O último comando executado foi o "aluno.reverse!", que irá retornar a *String* reversa, no caso "navetsE".

Ruby também possui outras características importantes, como a de seu compilador ser em código aberto e disponível no Github e possuir tipagem forte e dinâmica. A linguagem *Ruby* possui uma peculiaridade, que a torna ímpar comparada a outras linguagens, ela foi desenvolvida com foco nas pessoas, fornecendo uma sintaxe limpa e elegante, a qual facilita a leitura e entendimento do código até mesmo por pessoas iniciantes na área da computação (THOMAS, 2008), sendo requisito, além disso, apenas um bom entendimento da língua inglesa. Como pode ser observado na figura 2 contendo a definição do método "veralunosturma*"* da classe "ProfessoresController" (não mostrada na figura).

```
15      def veralunosturma
16          @professor = Professor.find(params[:id])
17          @turma = @professor.turmas.find_by_id(params[:idturma])
18          respond_with @turma
19      end
```

Figura 2 – Método "veralunosturma" da classe "ProfessoresController".

Seu criador tinha o objetivo de fazer uma linguagem que proporcionasse diversão ao programador, diminuindo as dificuldades presentes no desenvolvimento de *software* (RUBY, 2010).

2.2 FRAMEWORK RUBY ON RAILS

Ruby on Rails é um *framework* para desenvolvimento *web*, utilizando a linguagem *Ruby*, lançado ao público pela primeira vez em julho de 2004, por David Heinemeier Hansson. Segundo Monteiro (2012), "*Ruby on Rails* é um *framework* de desenvolvimento web otimizado para a produtividade sustentável e diversão do programador. Ele permite que você escreva código de maneira elegante, favorecendo convenção ao invés de configuração".

Rails surgiu da junção de vários outros *frameworks*, tornando-se assim um *meta-framework*. Entre os *frameworks* que compõem o *Rails* estão (FILHO, 2012):

- **Active Record:** é considerado um *framework* que contém uma camada de mapeamento objeto-relacional, entre a aplicação e o banco de dados;
- **Action Pack:** framework HTML (HyperText Markup Language), XML (eXtensible Markup Language), Javascripts para controle de regras de negócio;
- **Action Mailer:** *framework* recebimento de mensagens, capaz de realizar diversas operações apenas com chamadas de entregas de correspondência;
- **Active Support:** *framework* que contém coleções de classes e extensões de bibliotecas, consideradas úteis para uma aplicação em *Ruby on Rails*;
- **Active *WebServices*:** *framework* que provê uma maneira de publicar APIs (*Application Programming Interface*) que se comuniquem com o *Rails;*

Um outro atrativo do *framework* é que ele permite que as funcionalidades de um sistema possam ser implementadas de maneira incremental (ou seja, de forma gradual) por conta de alguns padrões e conceitos adotados. Por exemplo, não é necessário encontrar a posição ideal no código para um método, basta inseri-lo ao final do arquivo da classe atual. Isso tornou o *Rails* uma das alternativas para projetos e empresas que adotam metodologias ágeis de desenvolvimento de software. A arquitetura principal das aplicações criadas utilizando o *framework Rails* são desenvolvidas com base no padrão MVC (*Model-View-Controller*), padrão arquitetural no qual os limites entre seus modelos, suas lógicas e suas visualizações são bem definidos, sendo muito mais simples fazer um reparo, uma mudança ou uma manutenção, já que essas três partes de comunicam de maneira bem desacoplada (CAELUM, 2013). Na Figura 3 é possível verificar a estrutura dos diretórios criados no projeto *Rails* observada por meio do editor de texto Sublime Text 2, escolhido para o desenvolvimento do *software*.

Figura 2 – Estrutura da aplicação Rails. Fonte: Autoria Própria

Ao criar um projeto em *Rails*, ele irá gerar vários arquivos e diretórios destinados a estruruda MVC aos arquivos *javascript*, *Cascading Style Sheets* (CSS) e, também, os diretórios para testes. Os principais diretórios do *Rails* são descritos abaixo (CAELUM, 2013):

- **app:** se subdivide em outros diretórios no qual o programador passará a maior parte do tempo escrevendo sua aplicação;
- **app/controllers:** é o diretório responsável por abrigar os controladores da aplicação, que são as classes responsáveis por atender requisições e gerar respostas;
- **app/models:** é o diretório responsável por abrigar os modelos da aplicação, que são as classes responsáveis por interagir com banco de dados e conter a lógica de negócio;
- **app/helpers:** é o diretório responsável por abrigar arquivos que contém métodos para facilitar a implementação de lógicas a serem mostradas nas *views*;
- **app/views:** é o diretório responsável por abrigar a telas da aplicação;

- **app/assets:** diretório para os *assets*; ele irá disponibilizar todos os arquivos (imagens, *javascript* e css) dinamicamente para o *browser*;
- **config:** diretório para realizar toda a configuração da aplicação;
- **db/migration:** possui todos os arquivos responsáveis por gerar as tabelas no banco de dados a partir dos modelos da aplicação;

O *framework RoR* foi escolhido para o desenvolvimento desta aplicação *web* pelos seguintes motivos:

- Utiliza a linguagem *Ruby*, a que interessa aos autores aprofundar os conhecimentos;
- Suporta nativamente a utilização do padrão Modelo Visão Controlador (MVC).
- Permite gerenciar o banco de dados da aplicação diretamente a partir das propriedades das classes.
- Permite desenvolver aplicações baseadas em comportamentos.
- Possui diversas outras extensões ativamente suportadas.

Modelo Visão Controlador, ou *Model View Controler* (MVC), é um padrão de arquitetura de sistema, ou *Design Pattern*, que isola a modelagem, a interface com usuário e a lógica de um sistema (GAMMA, 2002). A camada de modelagem compreende os objetos que representam os dados utilizados no sistema e define as relações entre cada tipo de objeto. A camada de visão define como cada modelo é visualizado pelo usuário do sistema de acordo com o propósito. A camada de controle é responsável por manipular as requisições *Hypertext Transfer Protocol* (HTTP) da aplicação, ou seja manipular os modelos, as visões e validação, filtragem de dados e processamento da lógica de negócio.

O *RoR*, especificamente, permite que o banco de dados seja facilmente gerenciado, definido e mantido diretamente por meio do código da camada de modelo da aplicação, sem necessidade de gerenciar código de definições de banco de dados. Além disso, uma simples linha de comando no arquivo de configurações permite alterar o Sistema Gerenciador de Banco de Dados (SGBD) a ser utilizado (AKITA, 2006).

2.3 DESENVOLVIMENTO BASEADO EM COMPORTAMENTOS

BDD é uma técnica de desenvolvimento de sistemas que utiliza linguagem ubíqua, que possibilita o desenvolvedor focar nas razões pelas quais cada código deve ser programado ao invés de se preocupar com detalhes técnicos. A técnica consiste em descrever cada

funcionalidade específica do sistema em formato de estórias de usuário, durante a especificação de requisitos do sistema. Cada funcionalidade possui uma breve descrição contendo o ator, ou atores – como em um caso de uso – envolvido na funcionalidade, o objetivo que o ator deseja alcançar e as ações que o ator deve realizar para alcançar o objetivo. Em seguida, uma sequência de blocos de cenários que compõem cada funcionalidade descreve cada pequena parte do comportamento da aplicação. Cada cenário possui um título e depois uma sequência de pré-condições, uma sequência de ações e uma sequência de verificações. As pré-condições são definidas pelo comando "Dado"; as ações pelo comando "Quando"; e as consequências esperadas pelo comando "Então". O comando "E" repetido no início de diversas linhas repete o comando anterior, ou seja, se for utilizado depois de um "Quando", repete o "Quando" e, assim, sucessivamente (WYNNE, 2011).

2.3.2 GHERKIN

Estórias de usuário são uma forma mais ágil de se entender o que se deve fazer. Trata-se de resumir em uma sentença simples, sem pensar em muitos detalhes, o que o cliente realmente deseja como funcionalidades do *software* a ser desenvolvido. Um exemplo pode ser visto no Quadro 1.

```
Funcionalidade: "Descrição da funcionalidade"
Para atingir um determinado objetivo
Como uma pessoa que exerce um determinado papel no sistema
Desejo tal funcionalidade
```

Quadro 1 - Código de funcionalidade escrita em Gherkin.

Gherkin é uma linguagem comum que quebra Estórias em cenários. Cada cenário específico aborda uma situação diferente na estória. É possível notar que nesta linguagem não é abordada nenhuma característica técnica, como a definição de estruturas de dados e diagramas de entidades e relacionamentos. Será empregada uma linguagem natural estruturada próxima ao que o cliente utiliza. No quadro 2, pode-se verificar um *template* básico para definir um cenário (URUBATAN, 2009). O parâmetro "Cenário" determina o nome do cenário, o parâmetro "Dado" determina a pré-condição para que o cenário ocorra, o parâmetro "Quando"

determina a ação que acontece no cenário, e o parâmetro “Então” determina a verificação que é feita ao final do cenário que define que ele está correto.

`Cenário`: “Nome do Cenário”
`Dado` que um contexto se crie (pré-condição)
`Quando` uma interação acontece (ação)
`Então` pode-se extrair uma evidência (verificação)

Quadro 2 – Código de cenário escrito em Gherkin.

2.3.1 CUCUMBER

A ferramenta utilizada neste TCC para realizar o BDD é o *Cucumber* que permite compilar comportamentos – em formatos de funcionalidades de sistema na forma de texto simples – na forma de testes automatizados utilizando a linguagem *Gherkin* (WYNNE, 2011). Os comportamentos devem ser o princípio de qualquer caso de uso do sistema e, assim que comportamentos são compilados, o *Cucumber* procura por funções de teste que se encaixem em suas descrições e retorna ao desenvolvedor uma mensagem de sucesso ou de erro descrevendo o que falta ser desenvolvido no código do sistema para que cada comportamento funcione como definido (NORTH, 2006).

A instalação do *Cucumber* pode ser realizada adicionando a linha *gem ‘cucumber-rails’* no arquivo *Gemfile* do projeto. Logo em seguida será necessário executar o comando *bundle install* que descarregará da *internet* todas as dependências de *frameworks.* Feito isso sua instalação poderá ser realizada com o comando: *bundle exec rails g cucumber:install.* O projeto estará pronto para uso, com isso o suporte ao *Cucumber* será adicionado à aplicação, bem como uma pasta “*/features”*. A pasta *features* será onde estarão todos os arquivos “*.feature*”, cada um desses arquivos, escritos na linguagem *Gherkin* que poderá ser visto com mais detalhes no apêndice deste TCC. Cada *feature* inicia com a descrição de sua funcionalidade em forma de estória de usuário (URUBATAN, 2009), conforme ilustrado no Quadro 3 para um exemplo de funcionalidade chamada “Gerenciar usuários”.

Funcionalidade: Gerenciar usuários
A fim de gerenciar usuários
Os visitantes devem ser capazes de

Cadastrar-se e editar o seu próprio conteúdo

Quadro 3 – Funcionalidade "Gerenciar usuários".

Em seguida, adicionam-se os cenários possíveis que se aplicam a essa funcionalidade. Nesse caso, pode-se pensar nas situações que podem ocorrer para gerenciar os usuários. Posteriormente, o desenvolvedor poderá adicionar mais situações que não foram previstas inicialmente e situações que estão causando comportamentos inesperados na aplicação. O Quadro 4 apresenta um exemplo de cenário chamado "Adicionar novo usuário" para a funcionalidade "Geranciar usuários".

Cenário: Adicionar novo usuário
Dado que o visitante quer se inscrever
Quando o visitante clicar em inscrever
Então abra a página de inscrição

Quadro 4 – Cenário "Adicionar novo usuário".

Ao executar o comando **cucumber**, será exibida uma saída no console com o conteúdo apresentado no Quadro 5.

Funcionalidade: Gerenciar usuários
A fim de gerenciar usuários
Os visitantes devem ser capazes de
Cadastrar-se e editar o seu próprio conteúdo.
Cenário: Adicionar novo usuário # features/manage_users.feature:6
Dado que o visitante quer se inscrever # features/gerenciar_usuarios.feature:7
Quando o visitante clicar em inscrever # features/ gerenciar_usuarios.feature:8
Então abra a página a página de inscrição # features/ gerenciar_usuarios.feature:9
1 scenario (1 undefined)
3 steps (3 undefined)
0m0.297s
You can implement step definitions for undefined steps with these snippets:

```
Dado /^que o visitante quer se inscrever$/ do
pending
end

Quando /^o visitante clicar em inscrever$/ do
pending
end

Então /^abra a página a página de inscrição$/ do
pending
end
```

Quadro 5 – Execução do comando "cucumber" no console com a funcionalidade e cenário descritos.

O texto em amarelo significa que os testes relativos a estas sentenças da funcionalidade (`step` na documentação do *Cucumber*) ainda não foram implementados. O código apresentado ao final da execução do comando (de "Dado" até "Então... end", em amarelo) poderá ser copiado em um outro arquivo com a extensão "*.rb"* que deverá conter o teste em si, como por exemplo:

```
Dado /^que o visitante quer se inscrever$/ do
#comando que irá simular a visita do cliente à página principal, onde temos um link com o seguinte conteúdo: "Quero me cadastrar"
get '/'
end
```

Quadro 6 – Código do arquivo que testa se a primeira sentença do cenário é verdadeira.

Ao executar o comando **cucumber** novamente, a saída ilustrada no Quadro 7 será apresentada.

```
Cenário: Adicionar novo usuário # features/manage_users.feature:6
Dado que o visitante quer se inscrever # features/gerenciar_usuarios.feature:7
Quando o visitante clicar em inscrever # features/ gerenciar_usuarios.feature:8
Então abra a página a página de inscrição # features/ gerenciar_usuarios.feature:9
```

Quadro 7 – Execução do commando "cucumber" com um teste resultando em verdadeiro.

A linha em verde indica que o teste passou. O modo de se trabalhar com o *Cucumber* é, assim como o TDD, escrever o teste, dentro de um *step*, executar e vê-lo falhar. Ao falhar, a linha será apresentada em vermelho. Deve-se programar somente o necessário para o teste passar (ficar verde), e passar imediatamente para o próximo *step*. Ao realizar todos os *steps* aquela funcionalidade estará completa de acordo com a estória de usuário, que foi descrita de acordo com a necessidade e aval do cliente.

2.4 EDUCAÇÃO FUNDAMENTAL

A educação fundamental é a segunda etapa da educação básica no Brasil. A educação básica propõe garantir a formação do cidadão a fim de conhecer seus direitos e deveres na Constituição, a educação envolve o ser humano em suas relações individuais, civis e sociais. As Diretrizes Curriculares Nacionais – DCN – (CNE/CEB, 1998) orientam as escolas nos sistemas de ensino, organização, articulação, desenvolvimento e avaliação das propostas pedagógicas. As diretrizes definem a Educação Fundamental como obrigatória para todos brasileiros, seus direitos, e é portanto dever do órgão do Estado responsável a garantia deste direito. As DCN definem que deve ser garantido o acesso dos alunos à Base Nacional Comum e sua Parte Diversificada – conteúdos complementares definidos por cada sistema de ensino ou estabelecimento, de acordo com características regionais – incorporados ao paradigma curricular relacionando a Vida Cidadã com as Áreas do Conhecimento de acordo com as Leis de Diretrizes e Bases (LDB) da Educação Nacional (BRASIL, 1996). Os conteúdos mínimos de cada área do conhecimento podem ser redefinidos anualmente pelo MEC, cada órgão municipal ou estadual responsável pelo Ensino Fundamental deve então garantir o acesso a esses conteúdos nos estabelecimentos de suas dependências que podem utilizar de quaisquer outros meios nas suas propostas pedagógicas para garantir ao máximo o acesso dos alunos à educação como estabelecido nas DCN.

2.4.1 CICLOS DE APRENDIZAGEM

A implantação dos ciclos de aprendizagem ocorreu com objetivo de que o cotidiano pedagógico das escolas municipais passasse por uma transformação em resposta às transformações que

ocorrem na sociedade (CURITIBA, 1999). Este projeto propôs uma estrutura composta de quatro ciclos no ensino fundamental, substituindo a divisão de 1ª à 8ª série.

O Ciclo I foi, inicialmente, estipulado a ter dois ou três anos (também chamados de etapas) de duração, com prioridade para crianças a partir de sete anos de idade, ou a completar, no ano civil em curso com duração de dois anos. No caso de crianças a completar seis anos a duração do ciclo é de três anos e o número máximo de alunos por turma deste ciclo é de trinta alunos. O Ciclo II tem duração de dois anos letivos para alunos a completar nove anos de idade até dez anos. Os Ciclos III e IV possuem ambos dois anos letivos de duração para as idades relativas aos anos seguintes dos ciclos anteriores. Estes três últimos ciclos podem ter até trinta e cinco alunos por turma (CURITIBA, 2005).

No caso da Escola Municipal Walter Hoerner, objeto de investigação deste projeto, somente os dois ciclos iniciais são organizados por não ofertar o Ensino Fundamental completo. Todas as escolas municipais foram determinadas a implantar totalmente o sistema de Ciclos de Aprendizagem com prazo máximo para o final do ano de 2010, portanto todas já estão com este sistema implantado, sem exceção (CURITIBA, 1999).

O Sistema de Ciclos trouxe consigo outras modificações nas práticas pedagógicas e qualificação de educadores, sendo feita uma revisão do **Currículo Básico** das Escolas da Rede e a contratação no ano de 1999 de trezentos e trinta e quatro profissionais para atuarem como corregentes e apoio pedagógico às escolas. Inclusive, até a data atual, a SME de Curitiba vem sido reconhecida e tratada como objeto de estudo de diversos pesquisadores pela qualidade do projeto realizado, que é extremamente bem quisto no plano de desenvolvimento administrativo e pedagógico para os educadores da rede.

Transformações no âmbito de processos de avaliação também foram significativas com este projeto, com o ideal de acompanhar as aquisições sucessivas de cada aluno ao longo do processo educativo. Possibilitando, assim, superar das dificuldades ainda no decorrer do ensino-aprendizagem. Neste sistema, o aluno pode ter uma **Progressão Simples** na qual o aluno seguirá normalmente de um ciclo a outro, ou uma progressão com necessidade de **Apoio Pedagógico** em que é necessário um plano pedagógico de apoio para cada caso específico. Com isto, na estrutura de **Ciclos de Aprendizagem**, o estudante somente permanecerá – por no máximo mais um ano - no ciclo de origem caso não atinja a frequência mínima de 75% da carga horária total letiva do ciclo (CURITIBA, 1999).

2.4.2 ENSINO FUNDAMENTAL DE NOVE ANOS

Pais ou responsáveis têm o dever de matricular crianças a partir de seis anos de idade no ensino fundamental sem a diminuição dos recursos médios por aluno na rede pública (BRASIL, 2005). Deliberada, então, em 2008, no município de Curitiba, pelo Conselho Municipal de Educação, a matrícula, nas escolas da Rede Municipal de Ensino, de crianças de seis anos de idade na primeira etapa do Ciclo I do Ensino Fundamental – que passa a vigorar somente com duração de três anos letivos, enquanto o Ensino Fundamental completo passa a ter nove anos de duração (CURITIBA, 2008). Transformação está de acordo com o Plano Nacional de Educação (PNE) de 2001 que não possuía prazo, mas estipulava a implantação deste sistema à medida que o ensino de sete a quatorze anos fosse universalizado (BRASIL, 2001).

2.4.3 AVALIAÇÃO DE DESEMPENHO ACADÊMICO

A avaliação serve para o professor acompanhar e melhorar o desempenho dos alunos, diagnosticar resultados e atribuir-lhes valores e estudar e interpretar como anda a aprendizagem e seu próprio trabalho, dando-lhe condição para tomar decisões para o aperfeiçoamento da aprendizagem. E quando preciso também serve para reorganizar os conteúdos e métodos de ensino. Para cumprir a sua finalidade educativa a avaliação deverá ser contínua, permanente e cumulativa. O acompanhamento do processo de avaliação da série, ciclo, grau ou período cabe ao órgão indicado pelo Regimento Escolar e é dele a responsabilidade de debater e analisar todos os dados referentes às avaliações (PARANÁ, 1999).

Nas avaliações são utilizados técnicas e instrumentos diversificados que visam à interdisciplinaridade e a multidisciplinaridade dos conteúdos, considerando a capacidade individual, o desempenho do aluno e sua participação nas atividades realizadas. Para o ensino da Educação Física e de Arte, devem ser adotados procedimentos diferenciados de avaliação visando ao desenvolvimento formativo e cultural do aluno.

2.4.4 AVALIAÇÃO NA ESCOLA MUNICIPAL WALTER HOERNER

A ideia do projeto surgiu em uma reunião com a Equipe Pedagógica e Administrativa (EPA) da Escola Municipal Walter Hoerner que foi realizada com o intuito de encontrar uma necessidade da escola de um sistema de informação baseado em computador que auxiliasse a escola a realizar algum tipo de avaliação de desempenho acadêmico de seus alunos. Para isto,

primeiramente, a discussão foi direcionada ao esclarecimento sobre os métodos utilizados por esta escola para avaliar seus estudantes, os objetivos de cada avaliação e por que cada método existia. A equipe da escola constatou que a Secretaria Municipal da Educação de Curitiba já desenvolveu um sistema para avaliação dos alunos, feito totalmente com a ferramenta *Microsoft Excel*, sendo composto por algumas tabelas, que devem ser preenchidas pela escola a cada semestre, com resultados de provas de trinta questões de cada disciplina, de acordo com a etapa e o ciclo em que cada aluno está.

Entretanto, do ponto de vista da equipe pedagógica e administrativa da escola, este método de avaliação não permite que casos de alunos com dificuldade de aprendizado em algum conteúdo ou disciplina específica, assim como casos de turmas que precisam de um redimensionamento de prática de ensino, sejam detectados a curto prazo. O acompanhamento e recuperação desses casos exigem um esforço tal que diminui, consideravelmente, o tempo dos funcionários da escola para outras atividades relacionadas ao ensino – como a preparação de aulas e materiais de ensino. Este esforço se deve a quantidade de papéis gerada mensalmente pelo processo e a dificuldade de se identificar os casos a serem trabalhados.

Com base neste problema, a EPA desenvolveu um sistema parecido com o sistema de avaliação da prefeitura, que é utilizado paralelamente a este último pelos professores e pela própria EPA de maneira a otimizar o tempo de resposta aos casos descritos acima. O sistema consiste em uma tabela em um arquivo eletrônico em formato do *Microsoft Word* – Anexo A. A diferença é que deve ser realizado mensalmente e, apesar dos conteúdos também serem retirados da mesma fonte do programa computacional da prefeitura, eles foram distribuídos em três trimestres por ano – Anexo B – e conjuntos de conteúdos foram definidos para cada um deles. Deste modo, todos os conteúdos específicos de cada etapa dos ciclos são avaliados no período correspondente. Assim, o professor tem a liberdade de escolher os conteúdos de cada disciplina que avaliará mensalmente, pois ele pode avaliar interdisciplinarmente, desde que os conteúdos predefinidos pela EPA sejam avaliados no trimestre correspondente. Finalmente, os resultados das avaliações eram analisados, discutidos, as devidas providências eram tomadas para cada caso que não atendesse os critérios pré-estabelecidos – Anexo C – e, enfim, eram arquivados em papel em escaninhos na escola para futuras consultas.

Portanto há uma necessidade, não somente desta escola, em fazer este processo de reação às avaliações de maneira mais eficiente possível. E neste caso entra o *software* desenvolvido neste TCC, que propõe substituir o sistema em papel atual. E ainda em caso de sucesso após um teste de aceitação na escola em questão, poderá ser extrapolado para as demais

escolas da Prefeitura de Curitiba ou até, quem sabe, escolas de outros municípios, estados ou até escolas particulares interessadas.

3 METODOLOGIA

Este capítulo comtempla a metodologia seguida para o desenvolvimento do software objeto final deste TCC, inclusive as funcionalidades especificadas e suas descrições.

3.1 VISÃO GERAL DO DESENVOLVIMENTO

O primeiro passo para o desenvolvimento do *software* objeto deste TCC é a definição dos comportamentos básicos do sistema que são utilizados para desenvolver as primeiras funcionalidades do *software*. Esses comportamentos devem seguir o formato de estórias de usuário, descrevendo as funcionalidades do sistema uma a uma de acordo com o ponto de vista do usuário. Também, deve-se utilizar palavras chave da linguagem *Gherkin* que podem ser interpretadas pelo *Cucumber*, sendo capaz de referenciar quais objetos e métodos (próximos as palavras chave) em cada sentença não estão consistentes no código do sistema, ou seja, não estão presentes nos arquivos do projeto.

O segundo passo envolve a realização de adaptações que consistem em substituições de palavras genéricas em cada cenário por palavras que representem objetos e métodos reais no sistema considerado. Porque ao executar o *Cucumber*, estes objetos especificados nos cenários são criados durante a execução dos testes para representar objetos que existirão no sistema em produção.

A seguir, é definido o padrão de interface gráfica do sistema de acordo com uma identidade a ser determinada. Então, são elaborados os casos de teste de acordo com os comportamentos definidos anteriormente e para que as funcionalidades do *software* comecem a ser programadas. O código será gerenciado por meio de uma ferramenta de compartilhamento e versionamento de código, chamada Github, sendo levado em consideração o objetivo do projeto e as opções de licenciamento das ferramentas. Por fim, o sistema será implantado na web em uma plataforma que permita que o processo de implantação ocorra de maneira simples, segura e eficiente. A melhor candidata, até o momento, para ser usada como plataforma para implantação da aplicação é o *Heroku* (HEROKU, 2007). As principais razões para utilizar esta plataforma são:

- ela possui uma extensão no *RoR* que permite ser usada por meio de uma simples linha de comando sempre que alguma versão nova do código for enviada para o sistema de versionamento e, então, definida como versão de produção.

- é uma plataforma gratuita, mesmo que com limitações de memória e processamento.
- integra-se ao GitHub, utilizado neste projeto.

Finalmente, o sistema só poderá ser considerado um artefato pronto para uso aberto ou comercial quando possuir uma estrutura simples de autorização e autenticação de usuários. Para isso será necessário o desenvolvimento das funcionalidades necessárias para tanto, porém como prova de conceito para a realização deste TCC, não será necessário a não ser que haja tempo ao final do projeto.

3.2 ESTÓRIAS DE USUÁRIO

O primeiro passo para o desenvolvimento de *software* baseado em comportamento é a elaboração das estórias de usuário. As estórias de usuário deste capítulo foram descritas pela EPA de uma escola no município de Curitiba. Após a descrição dessas estórias, foram feitas as adaptações necessárias, como, por exemplo, substituir as palavras “uma turma” por “turma 4A” – para que houvessem as palavras chave a serem interpretadas pelo *Cucumber* em cada estória.

A lista de estórias está em ordem de importância e relação de dependência das funcionalidades de acordo com a EPA entrevistada, por exemplo, é necessário que seja possível gerenciar as turmas e seus objetos relacionados por meio do sistema antes de poder utilizar o *software* para avaliar os alunos da turma em questão. Outras estórias bastante úteis podem ser adicionadas em ciclos de desenvolvimento posteriores após a aprovação pelas partes interessadas do resultado do sistema como descrito pelas estórias já feitas.

3.2 FUNCIONALIDADES

As funcionalidades definidas para o *software* resultantes da análise feita junto com a EPA, estão listadas na tabela a seguir. Nas seções a seguir são detalhadas estas funcionalidades e apresentados os cenários respectivos.

Funcionalidade	Descrição
Gerenciar Turmas	Os membros da EPA devem ser capazes de cadastrar turmas únicas a cada ano, com seus respectivos alunos e professores.

Gerenciar EPA	Os membros da EPA devem ser capazes de cadastrar novos membros da EPA, além de editarem seus próprios dados pessoais no sistema.
Preparar Avaliações	Os membros da EPA devem ser capazes de registrar os conteúdos a serem lecionados e avaliados para cada disciplina de cada etapa de cada ciclo a serem definidos pelo MEC e adaptados pela EPA conformem acharem necessário.
Realizar Avaliação	Os professores devem ser capazes de visualizar e editar as avaliações realizadas para cada conteúdo para cada aluno em cada disciplina que leciona suas turmas.
Visualizar Avaliações	Os membros da EPA devem ser capazes de visualizar qualquer avaliação realizada por qualquer professor da escola para tomarem as respectivas ações pedagógicas.

Tabela 1 – Funcionalidades do software e suas descrições.

Essas funcionalidades foram, então, transformadas e adicionados detalhes como nomes de objetos referentes a cada classe em seus cenários, para que o *Cucumber* fosse capaz de interpretá-las. As funcionalidades do sistema estão situadas no apêndice deste TCC.

3.2.1 GERENCIAR TURMAS

Funcionalidade: Gerenciar Turmas

A fim de gerenciar as turmas da escola

Os membros da EPA devem ser capazes de

Cadastrar dados de turmas

Cenário: Adicionar nova turma

Dado que uma turma existe durante um ano letivo

Quando o membro da EPA clicar em cadastrar nova turma

Então abra a página de cadastro de nova turma

Cenário: Adicionar novo aluno

Dado que o aluno está matriculado na escola
Quando o membro da EPA clicar em cadastrar novo aluno
Então abra a página de cadastro de novo aluno

Cenário: Adicionar novo professor
Dado que o professor leciona na escola
Quando o membro da EPA clicar em cadastrar novo professor
Então abra a página de cadastro de novo professor

Cenário: Matricular aluno em turma
Dado que o aluno estuda em uma turma
Quando o membro da EPA clicar em matricular aluno em turma
Então abra a página da turma
E mostre o nome do aluno na turma

Cenário: Assinalar professor da turma
Dado que um professor ensina uma turma
Quando um membro da EPA clicar em assinalar professor da turma
Então abra a página da turma
E mostre o nome do professor na página

3.2.2 GERENCIAR EPA

Funcionalidade: Gerenciar EPA
A fim de gerenciar os membros da EPA da escola
Os membros da EPA devem ser capazes de
Cadastrar novos membros da EPA
E editar seus próprios dados

Cenário: Adicionar novo membro da EPA
Dado que um novo membro da EPA trabalha na escola
Quando o membro da EPA clicar em cadastrar novo membro da EPA
Então abra a página de cadastro de novo membro da EPA

Cenário: Editar dados de membro da EPA

Dado que o membro da EPA deseja alterar seus dados

Quando o membro da EPA clicar em editar dados

Então abra a página de editar seus próprios dados

3.2.3 PREPARAR AVALIAÇÕES

Funcionalidade: Preparar avaliações

A fim de preparar as avaliações da escola

Os membros da EPA devem ser capazes de

Cadastrar os conteúdos das disciplinas de cada ano

Cenário: Adicionar nova disciplina

Dado que uma disciplina será ministrada em um ano de ensino

Quando o membro da EPA clicar em cadastrar nova disciplina

Então abra a página de cadastro de nova disciplina

Cenário: Adicionar novo conteúdo de uma disciplina

Dado que um conteúdo de uma disciplina deve ser avaliado em um ano de ensino

Quando o membro da EPA clicar em cadastrar novo conteúdo

Então abra a página de cadastro de novo conteúdo

3.2.4 REALIZAR AVALIAÇÃO

Funcionalidade: Realizar avaliação

A fim de realizar as avaliações dos alunos da turma

Os professores devem ser capazes de

Visualizar e editar as avaliações dos alunos de suas turmas

Cenário: Visualizar turmas

Dado que o professor ensina uma turma

Quando o professor clicar em visualizar turmas

Então abra a página de turmas e liste as turmas do professor

Cenário: Visualizar alunos e disciplinas de turma

Dado que uma turma possua alunos e disciplinas

Quando o professor clicar na turma

Então abra a página da turma e liste os alunos e disciplinas da turma

Cenário: Visualizar conteúdos de disciplina

Dado que o professor leciona a disciplina a uma turma específica

Quando o professor clicar na disciplina

Então abra a página de conteúdos da disciplina

Cenário: Avaliar conteúdo

Dado que um professor ensinou um conteúdo à turma

Quando o professor clicar no conteúdo

Então abra a página de avaliar conteúdo

Cenário: Realizar avaliação de turma

Dado que um professor ensinou um conteúdo à turma

Quando o professor preencher a avaliação de cada aluno e clicar em Confirmar

Então abra a página de conteúdos da disciplina e os conteúdos avaliados devem estar marcados como avaliados

3.2.5 VISUALIZAR AVALIAÇÕES

Funcionalidade: Revisar avaliações

A fim de revisar as avaliações dos alunos da turma

Os professores e membros do EPA devem ser capazes de

Visualizar as avaliações dos alunos de suas turmas

Cenário: Visualizar avaliações de turma por disciplina

Dado que o membro do EPA ou professor deseja visualizar as avaliações de uma disciplina

Quando o membro do EPA ou professor clicar em visualizar avaliações da disciplina

Então abra a página de conteúdos e liste as avaliações para cada aluno da turma

Cenário: Visualizar avaliações de aluno

Dado que o membro do EPA ou professor deseja visualizar as avaliações de um aluno

Quando o membro do EPA ou professor clicar em visualizar avaliações do aluno

Então abra a página de boletim e liste as avaliações realizadas de cada disciplina e conteúdo

4 RECURSOS DE HARDWARE E DE SOFTWARE

Os usuários do sistema necessitam de um computador com acesso à internet e um navegador atualizado (como Chrome, Firefox e IE) instalado no sistema operacional para acessar o sistema. O sistema deverá ser executado em uma plataforma de serviços de servidores na nuvem, ou plataforma como serviço, cuja especificação é invisível a camada da aplicação ou aos usuários. Dentre as vantagens na escolha deste modelo para o servidor do sistema proposto estão a escalabilidade (PACHECO, 2009) e o custo para o projeto, sendo que diversos desses serviços são oferecidos gratuitamente para aplicações que utilizam poucos recursos, como é o caso deste projeto durante suas fases de desenvolvimento e aceitação.

Para o desenvolvimento do projeto, os autores utilizaram seus computadores pessoais com *hardware* padrão de PC comercial atual e com acesso à internet. Não houve nenhuma exigência especial sobre o *hardware* de desenvolvimento. Além disso, foram necessárias algumas ferramentas para a implementação do código, assim como o controle de versionamento, considerando que houve dois desenvolvedores, conforme descrito nas seções a seguir.

4.1 SUBLIME TEXT 2

O editor de texto escolhido para realizar o desenvolvimento do projeto foi o Sublime Text 2. Ele não é um *software* gratuito mas fornece uma versão de testes sem limite de tempo de uso, o que favoreceu a escolha por essa ferramenta. É um editor que possui muitos recursos interessantes, dentre eles os principais são: auxílio em auto completar o código, destaque de sintaxe (exibição de alto contraste) e, também, suporte à linguagem de desenvolvimento *Ruby on Rails* (BRUCE, 2012). Na figura 3 pode-se notar que sua interface é simples, uma vez que todos os arquivos, que constam no projeto podem ser rapidamente localizados no painel ao lado esquerdo, já o código fica centralizado e há, ainda, um mini mapa ao lado direito, que auxilia na navegação do código, caso ele seja muito grande (D'OLIVEIRA, 2011).

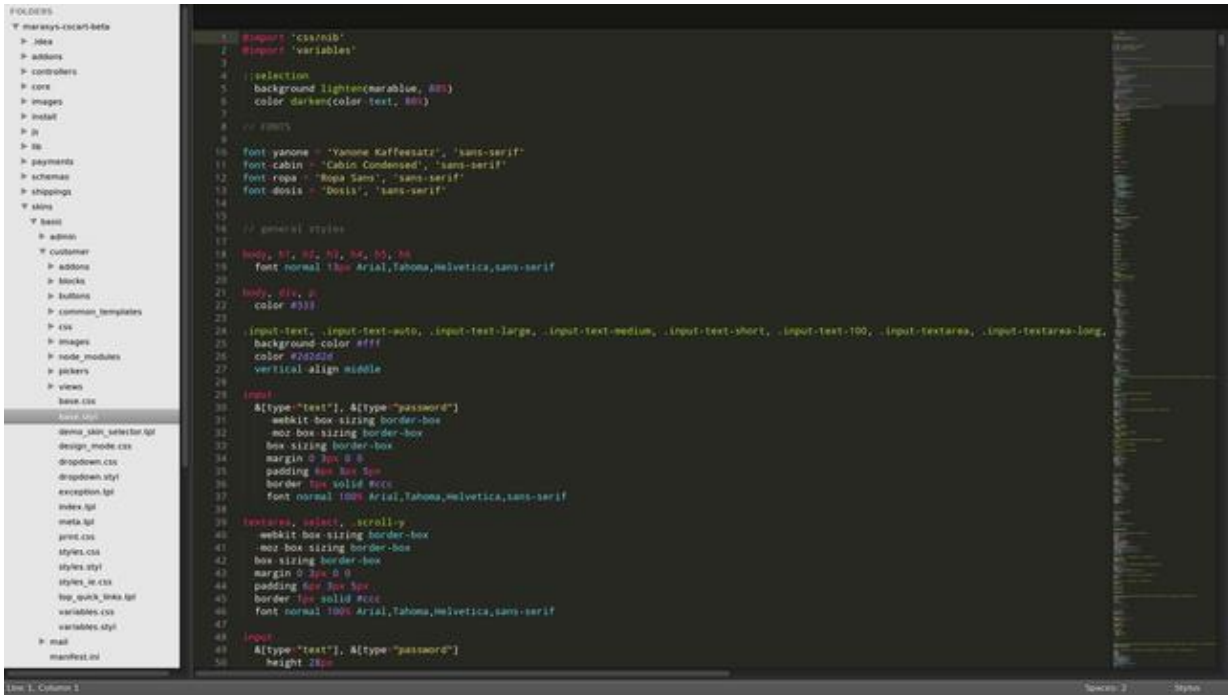


Figura 3 - Interface SublimeText2 (D'OLIVEIRA, 2011)

4.2 TWITTER BOOTSTRAP

O *Twitter Bootstrap* é uma coleção gratuita de ferramentas, que irão auxiliar e facilitar a criação da interface gráfica do sistema. O *Twitter Bootstrap* foi escolhido, pois facilita a criação da interface gráfica, disponibilizando diversos componentes customizáveis para realizar a criação de janelas, botões, formulários, etc.

O *Twitter Bootstrap* possui documentação técnica, que foi utilizada durante todo o processo de desenvolvimento da interface gráfica do sistema, pois ela fornece, grande número de orientações relacionadas à interface e usabilidade de uma aplicação (BOOTSTRAP, 2013). Basicamente, para utilizar este *framework* é necessário descarregá-lo, colocá-lo nas pastas do projeto e referenciar tanto seu arquivo CSS quanto seu arquivo *javascript* nas páginas web do projeto e, então, escolher o *layout* adequado para cada página ou deixar o básico. O *Twitter Bootstrap* facilita o desenvolvimento da interface gráfica do *software* reformulando a aparência dos componentes básicos HTML aproximando dos padrões atuais de *design web* sem a necessidade de implementação desses recursos básicos. A figura 5 apresenta o *layout* do site da NASA (seção de Código Aberto), produzido inteiramente com o auxílio da ferramenta *Twitter Bootstrap*.

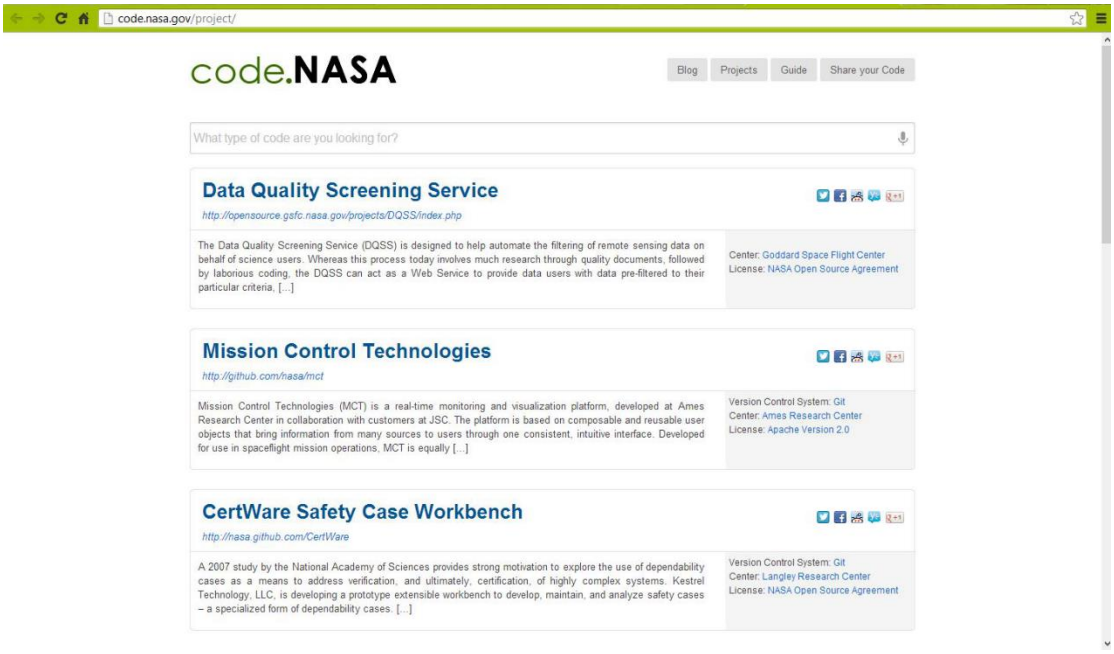


Figura 4 – Site da NASA utilizando *Bootstrap*. Fonte: http://open.nasa.gov/

4.3 GIT

O Git é um sistema de controle de versão – VCS (do inglês, *Version Control System*) e, também, um sistema de SCM (do inglês, *Source Code Management*) distribuído que está sendo cada vez mais usado (CHACON, 2013) e possui como grande vantagem para este projeto a facilidade de integração com GitHub (descrito na seção a seguir) e com o Heroku. O desenvolvedor inicial do Git foi Linus Torvalds (criador do *Linux*).

Um sistema de controle de versão tem a finalidade de gerenciar as diferentes versões dos artefatos de um projeto. Utiliza-se muito no desenvolvimento de software em que é preciso manter os artefatos do projeto documentados e organizados, fornecendo informações de tudo que foi e está sendo produzido até o momento.

Os principais comandos, que serão utilizados para realizar o controle de versão, integrados com o repositório online Git são (GIT, 2013):

- **git init** – comando que será executado somente uma vez e ficará responsável por criar um repositório Git dentro da pasta raiz do projeto. A partir desse comando será permitido realizar o gerenciamento dos arquivos contidos no diretório.
- **git status** – exibe o estado atual do repositório.

- **git add .** – inclui todos os novos arquivos e os arquivos modificados para área de publicação (área de *commits*).
- **git commit -m "Mensagem do *commit.*"** – publica os arquivos que foram selecionados pelo comando git add.
- **git clone git://github.com/git/projeto.git** – faz *download* de um repositório *online* do git.
- **git push** – envia todos os *commits* do computador local para um repositório *online* do git.
- **git pull** – baixa as últimas alterações de um repositório *online* do git para o seu repositório local.

4.4 GITHUB

O Github é um serviço de hospedagem distribuído, desenvolvido em *Ruby on Rails* para projetos que utilizam o Git. Assim, é utilizado como repositório *online* de códigos fonte para projetos de código aberto e privados para a versão paga.

Nele pode-se encontrar informações sobre todos os *commits* do projeto que o utiliza. É possível, ainda, incluir outras pessoas para auxiliar no desenvolvimento do projeto, possibilitando o trabalho simultâneo de todos os desenvolvedores, sem ocasionar conflitos (GITHUB, 2013). A tela inicial do Github pode ser vista na figura 5.

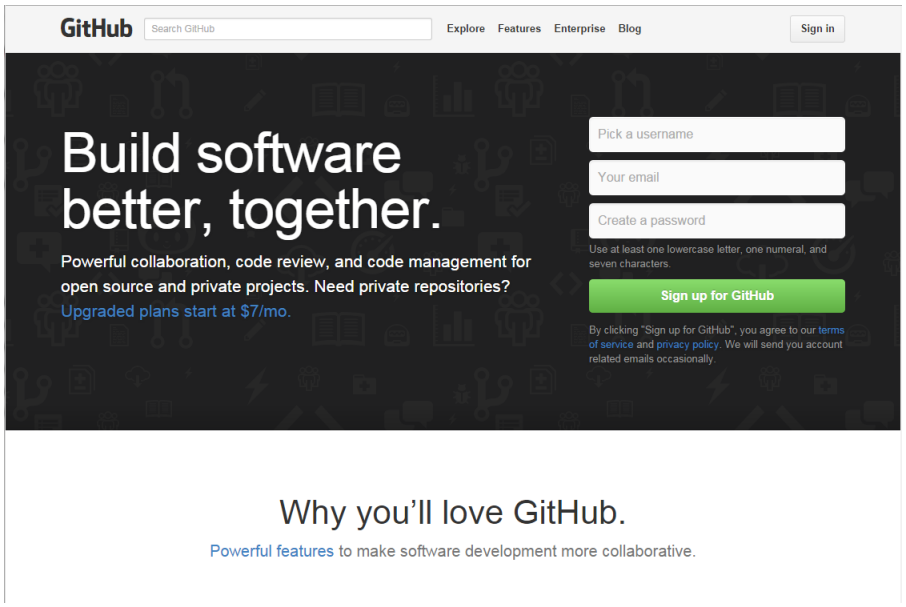


Figura 5 – Tela inicial do Github.

5 DESENVOLVIMENTO

Na seção 5.1 deste capítulo é descrito como foi desenvolvido o software neste TCC utilizando a técnica de BDD e, em seguida, em 5.2 é apresentado o estado atual do software com exemplos de utilização das interfaces desenvolvidas.

5.1 DESENVOLVIMENTO DO SOFTWARE

Primeiramente é realizada a instalação do *Ruby on Rails*, em seguida, a instalação das bibliotecas (*Rspec* e *Cucumber*) como descrito no capítulo 2. Então, uma pasta *features* foi criada dentro do projeto. Ela possuirá os arquivos "*.feature"*, que definem o comportamento da aplicação. Todos esses arquivos estão escritos em português e não em linguagem de programação *Ruby*; facilitando assim, a forma de comunicação entre o desenvolvedor e o cliente.

Esses arquivos seguem a sintaxe *Gherkin*, já explicada anteriormente, que pode ser observada na figura 6 que apresenta o arquivo "*realizar_avaliacao.feature"*. Neste arquivo há a definição, em português, de como a funcionalidade de realizar uma avaliação deve se comportar. Como pode ser observado, o arquivo define vários testes completos, com pré-condições, ações e verificações. Para que esses cenários tornem-se de fato testes executáveis, é necessário definir um *back-end* em *Ruby*.

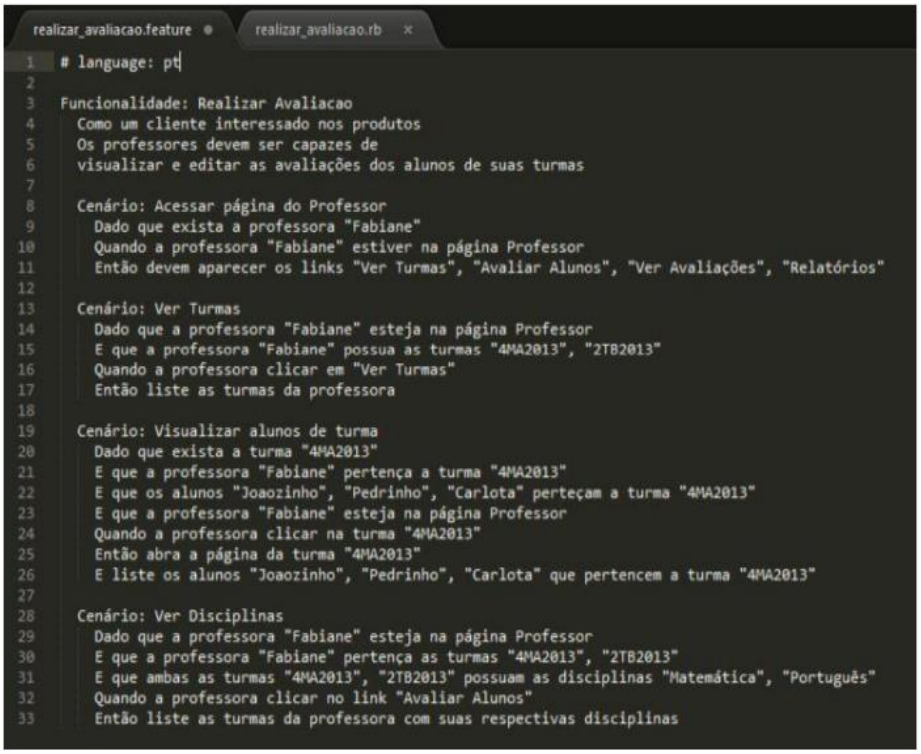


Figura 6 – Descrição da funcionalidade Realizar Avaliação.

Para executar os testes, o comando "*rake cucumber"* – figura 7 – deve ser executado. Após a execução do teste, o *Cucumber* irá imprimir uma mensagem – semelhante à figura 8 – dizendo que os cenários e os passos não estão definidos. Logo em seguida, ele imprime exemplos de código *Ruby* para a definição desses passos. Pode-se observar que palavras colocadas entre aspas são interpretadas como parâmetros a serem utilizados nos testes. Esses exemplos de código são muito úteis, pois servem de base para a criação de um arquivo de mesmo nome "*realizar_avaliacao"*, porém no formato "*.rb"*.

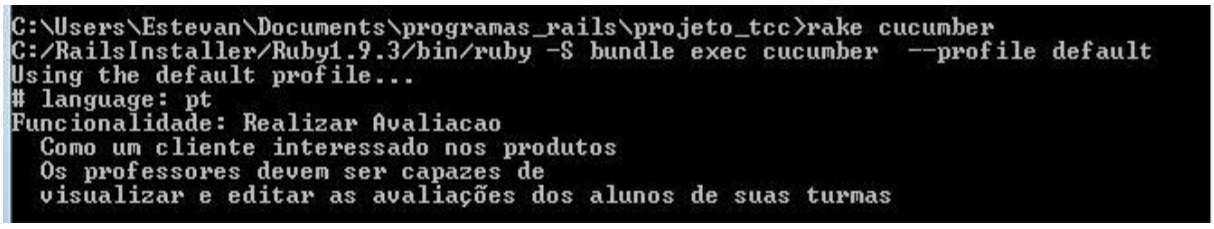


Figura 7 – Execução do teste através do comando rake cucumber.

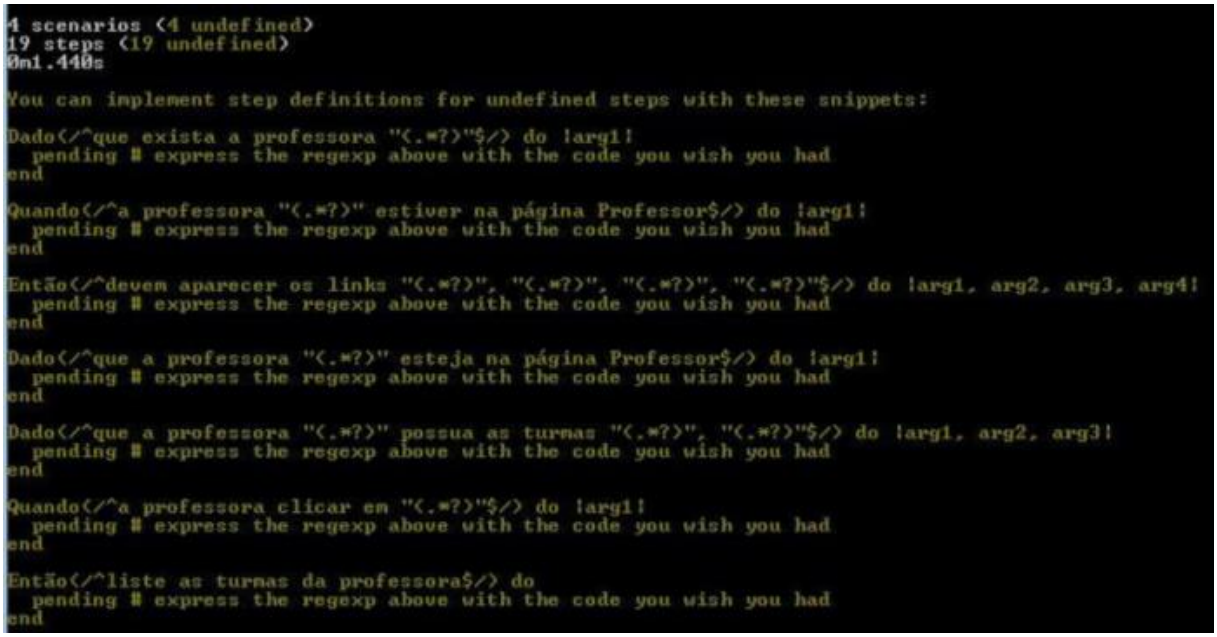


Figura 8 - Mensagem dizendo que os cenários e os passos não estão definidos.

A ideia é que os blocos de código *Ruby* "ensinem" o *Cucumber* a interpretar o texto em português, criando uma DSL (*Domain Specific Language*) para a criação dos testes de aceitação para o domínio da aplicação.

Vale lembrar que a aplicação possui vários arquivos ".feature" com seus respectivos executáveis no formato ".rb". Porém, como exemplificação, somente a funcionalidade principal (Realizar Avaliação) está sendo mostrada.

Para que seja possível implementar os testes de aceitação é necessário utilizar a biblioteca Capybara, que possibilita simular o acesso da aplicação ao automatizar um *browser*. Na figura 9, pode-se verificar o arquivo "realizar_avalicao.rb" depois de editado, ou seja, com todo o código que fará com que o teste funcione corretamente. O interessante é que, por meio dos comandos do Capybara, é possível navegar pela aplicação como se fosse um usuário comum, o que pode ser visualizado na imagem por meio do comando "visit" mostrado na linha 14. A primeira verificação do teste acontece na linha 22 desse mesmo arquivo, que por meio do comando (page.should have_content) do RSpec irá verificar se a página possui os conteúdos esperados. Com isso, as bibliotecas "capybara" e "rspec" auxiliam na escrita de todo o teste.

realizar_avaliacao.feature realizar_avaliacao.rb

```ruby

# Cenário: Acessar página do Professor

Dado(/^que exista a professora "(.*?)"$/) do |professora|

  Professor.create! nome: professora

end

Quando(/^a professora "(.*?)" estiver na página Professor$/) do |professora|

  p = Professor.find_by_nome(professora)
  #visit "/professores/#{p.id}"
  visit professor_path(p)

end

Então(/^devem aparecer os links (.*?)$/) do |links|

  list = links.split(", ").map{|s| s[1..-2] }
  list.each do |link|
    page.should have_content(link)
  end

end

# Cenário: Ver Turmas

Dado(/^que a professora "(.*?)" esteja na página Professor$/) do |professora|

  p = Professor.find_by_nome(professora)
  #visit "/professores/#{p.id}"
  visit professor_path(p)

end

Dado(/^que a professora "(.*?)" possua as turmas (.*?)$/) do |professora, turmas|

  p = Professor.find_by_nome(professora)

  list = turmas.split(", ").map{|s| s[1..-2] }
  list.each do |turma|
    Turma.create! nome: turma
    t = Turma.find_by_nome(turma)
    p.turmas << t
  end

end
```

Figura 9 – Arquivo de definição dos steps em Ruby depois de editado.

Após a criação de todo o teste, é necessário implementar código suficiente para que o teste passe. Ao executar os testes, todos os cenários não irão passar como mostrado na figura 10, pois nenhum código foi criado na aplicação.

```
Failing Scenarios:
cucumber features\realizar_avaliacao.feature:8 # Scenario: Acessar página do Professor
cucumber features\realizar_avaliacao.feature:13 # Scenario: Ver Turmas
cucumber features\realizar_avaliacao.feature:19 # Scenario: Visualizar alunos de turma
cucumber features\realizar_avaliacao.feature:28 # Scenario: Ver Disciplinas

4 scenarios (4 failed)
19 steps (5 failed, 14 skipped)
0m0.853s
```

Figura 10 – Resumo: Todos cenários falhando.

A continuação da execução dos testes pode ser verificada na figura 11, na qual se encontra de forma mais detalhada, o motivo do teste ter falhado. Nesse caso, verifica-se que o primeiro erro acusado foi da aplicação não possuir a entidade **Professor**, impossibilitando que um novo objeto do tipo **Professor** com o nome **Fabiane** fosse criado.

Figura 11 – Erros encontrados ao executar os testes.

Sendo assim, para que o *step* seja aprovado na aplicação, o modelo **Professor** deverá ser criado. Através do comando padrão do *Rails* para criar um novo modelo, foi criado o modelo **Professor** (figura 12), como requisitado ao executar o teste.

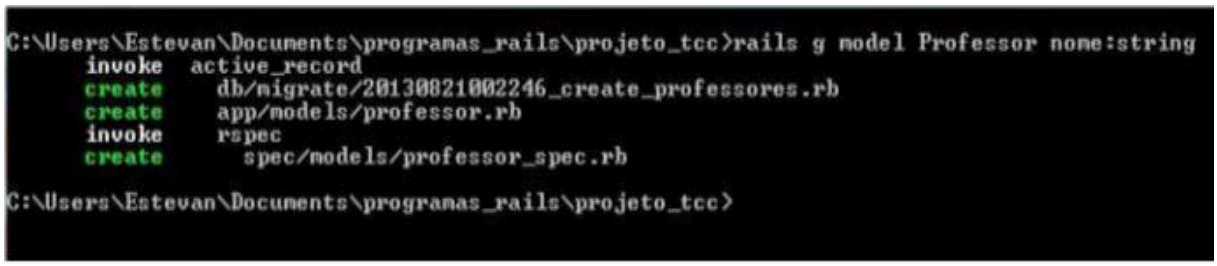


Figura 12 – Criação do modelo Professor.

Note que, ao executar novamente o teste, o *step* que havia falhado por não apresentar o modelo **Professor**, agora encontra-se na cor verde (figura 13). Ou seja, o *step* foi aprovado, pois de fato o modelo **Professor** passou a existir dentro aplicação. No entanto, outro erro foi

gerado (*undefined method professor_path*), assim, o *capybara* está tentando acessar uma *Uniform Resource Locator* (URL) e ela não está disponível na aplicação.

Figura 13 – Primeiro *step* do cenário *Acessar página do Professor*, aprovado.

Para que a URL seja criada dentro da aplicação será necessário alterar o arquivo *routes.rb* (figura 14), adicionando o conteúdo na linha 3. Serão criadas todas as URLs necessárias para o recurso **professores** seguindo o padrão *RESTful*, como pode ser verificado na figura 15.

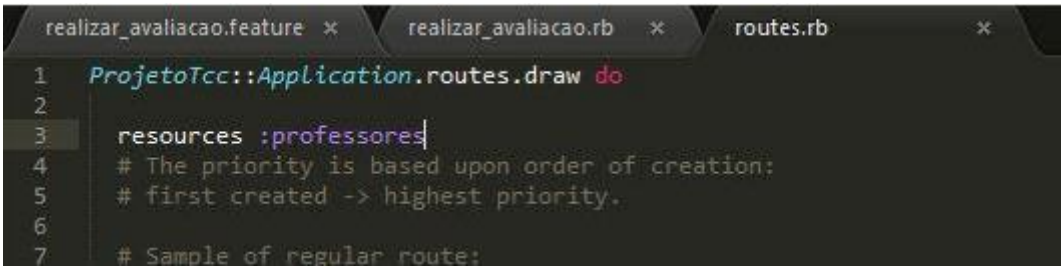


Figura 14 – Criação das rotas para os Professores.

Figura 15 – Listagem de todas as rotas geradas para o recurso Professores.

Executando o teste novamente outro erro foi gerado (figura 16), acusando que a controladora *ProfessoresController* não existe. O que pode ser facilmente contornado por meio do comando do *Rails*, para gerar a controladora requisitada, como pode ser visto na figura 17.

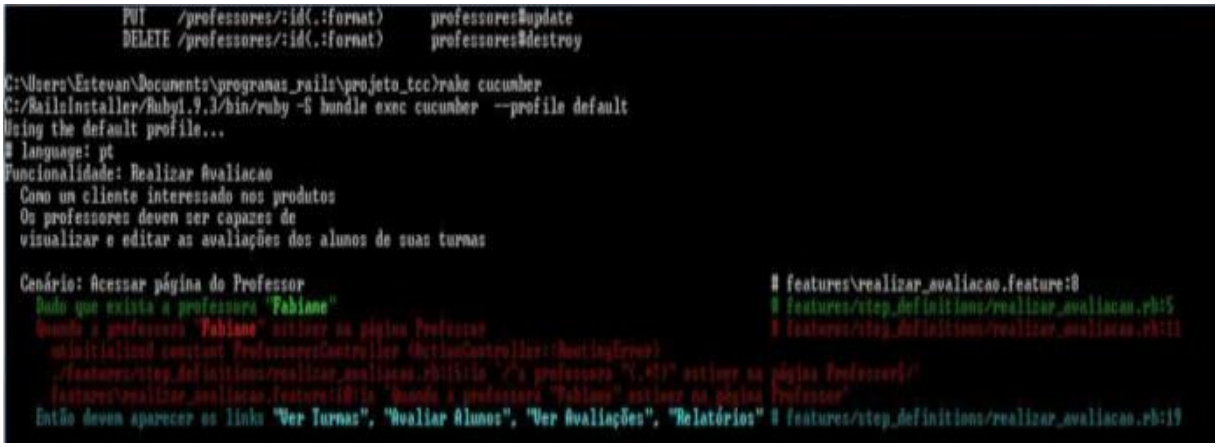


Figura 16 – Erro: Constante ProfessoresController não inicializada.

Figura 17 – Criação da controladora ProfessoresController.

Agora um novo erro surgiu (figura 18). Ao executar o teste ele conseguirá acessar a rota, logo em seguida irá encontrar a controladora. No entanto, não executará a *action show* e, com isso, conseguir *renderizar* a página do professor.

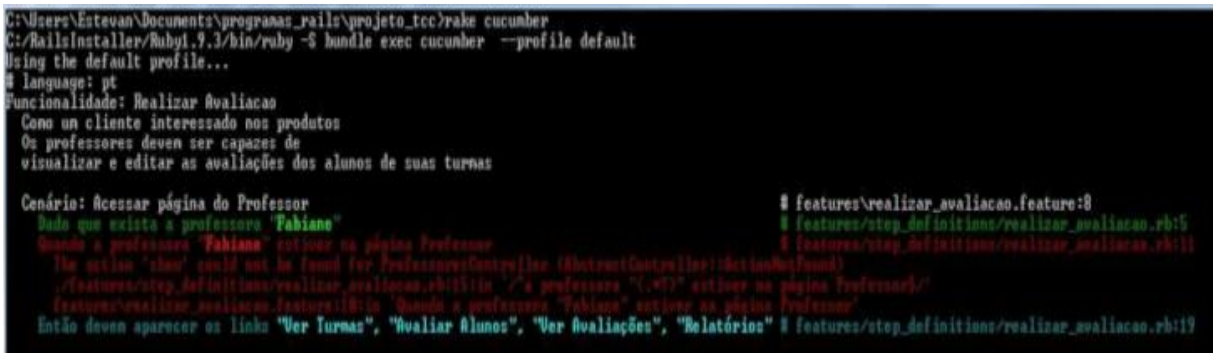


Figura 18 – Erro: Ação show não pode ser encontrada na controladora ProfessoresController.

Na figura 19 pode-se verificar que a *action show* foi criada dentro da controladora *ProfessoresController*. Porém, dois outros erros serão gerados. O primeiro, verificado na figura 20, é referente ao erro de não encontrar a visão *show*. Na figura 21, o arquivo *show.html.erb* foi criado e um segundo erro foi encontrado (figura 22). Nesse caso, o teste conseguiu realizar toda a requisição necessária, mas o teste irá procurar pelos *links* dentro da visão e não encontrará nada. Como pode ser visto na figura 23, o *html* com os *links* foi criado e os testes para o primeiro cenário ficaram *ok*.

```
class ProfessoresController < ApplicationController

    def show
        @professor = Professor.find(params[:id])
    end
end
```

Figura 19 – Criação ação show na controladora ProfessoresController.

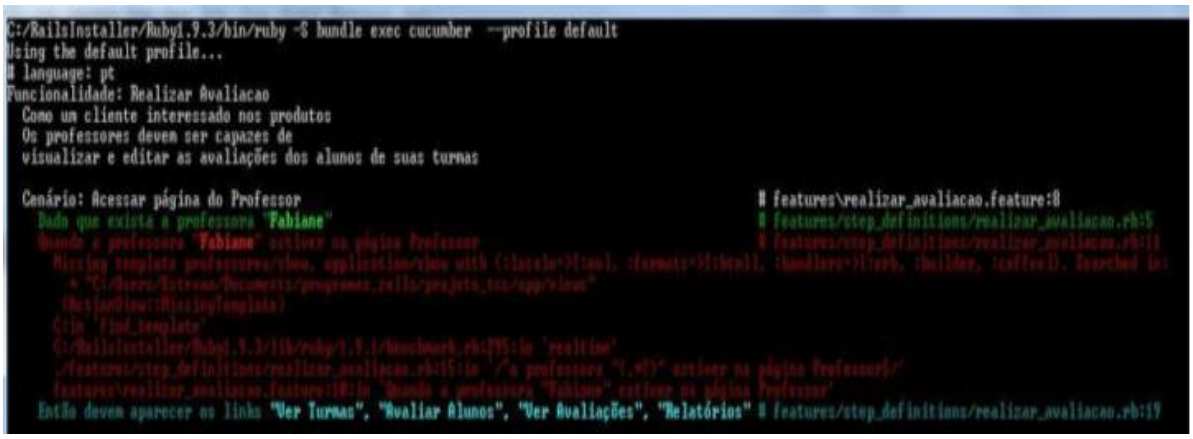


Figura 20 – Erro: Missing template professores/show.

Figura 21 – Criação da visão show.html.erb.

Figura 22 – Erro: Espera-se encontrar o texto Ver Turmas, Avaliar Alunos, Ver Avaliações, Relatórios.

Figura 23 – Criação dos links Ver Turmas, Avaliar Alunos, Ver Avaliações, Relatórios.

Ao criar os *links* dentro da visão "*show.html.erb"* e executar os testes (figura 24), todos os *steps* que pertencem ao cenário **Acessar página do Professor**, estarão verdes, ou seja, este cenário

está implementado. Agora, o próximo passo será *entrar* no cenário **Ver Turmas** e criar na aplicação todas as funcionalidades para que, assim, todos os *steps* sejam aprovados.

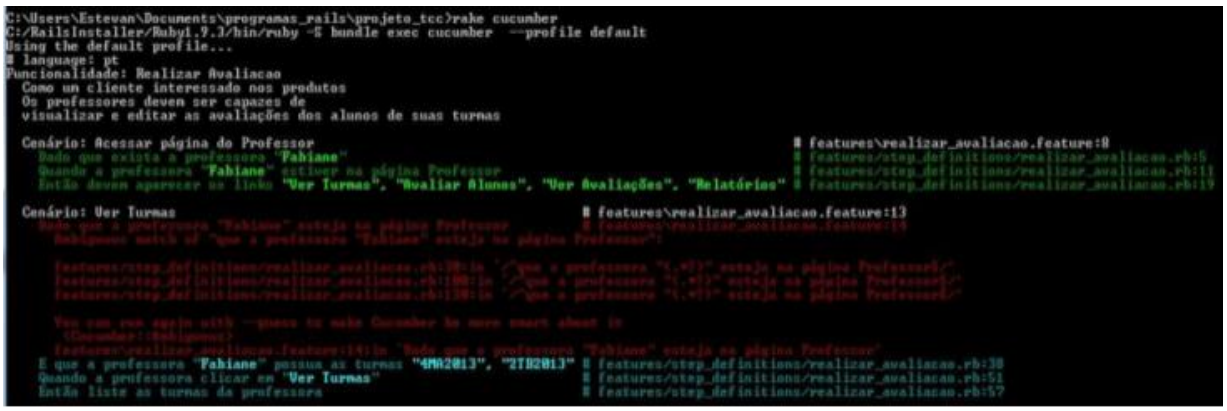


Figura 24 – Primeiro cenário com todos os passos aprovados.

Após esse exemplo detalhado, verificou-se que um cenário passou. Mas faltam ainda mais três para que a funcionalidade *Avaliar Alunos* esteja funcionando completamente e corretamente, como requisitada pelo cliente.

Figura 25 – Resumo: Apenas um cenário aprovado.

5.2 ESTADO ATUAL DO SOFTWARE

Nesta seção apresentamos uma seleção de telas com as principais funcionalidades do software desenvolvido. A primeira tela que o usuário irá encontrar ao acessar o sistema é a tela de *login* (figura 26), na qual o mesmo deverá inserir seu e-mail, senha e selecionar uma das opções (membro EPA ou professor) para conseguir entrar no sistema.

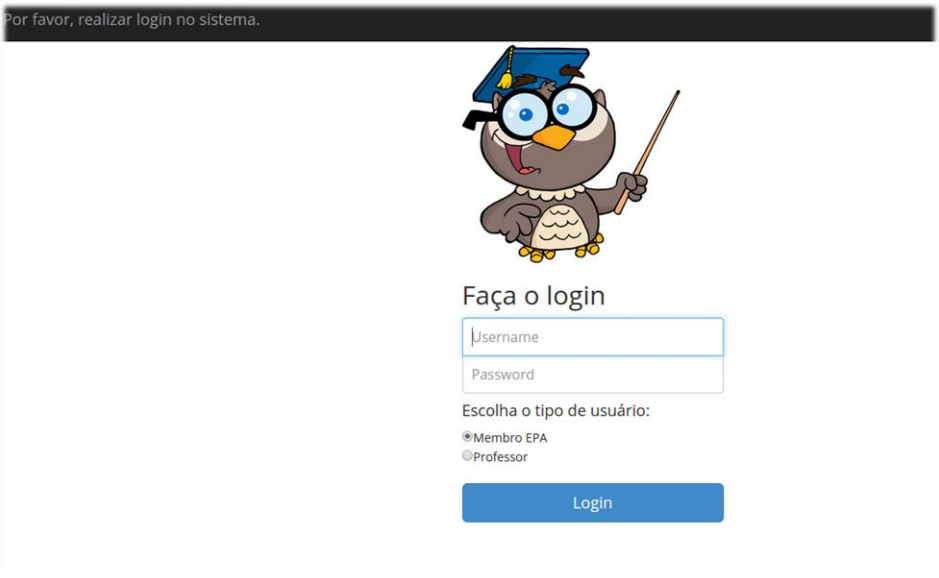


Figura 26 – Tela de *login* do software.

Após o usuário efetuar *login* no sistema, ele será redirecionado para a tela de menu principal. Caso o usuário seja um professor, será exibida uma tela com as opções que podem ser verificadas na figura 27; se o usuário realizar o *login* como membro EPA, ele irá acessar uma tela que apresentará diferentes funcionalidades (figura 28).

Figura 27 – Tela de menu de professores.

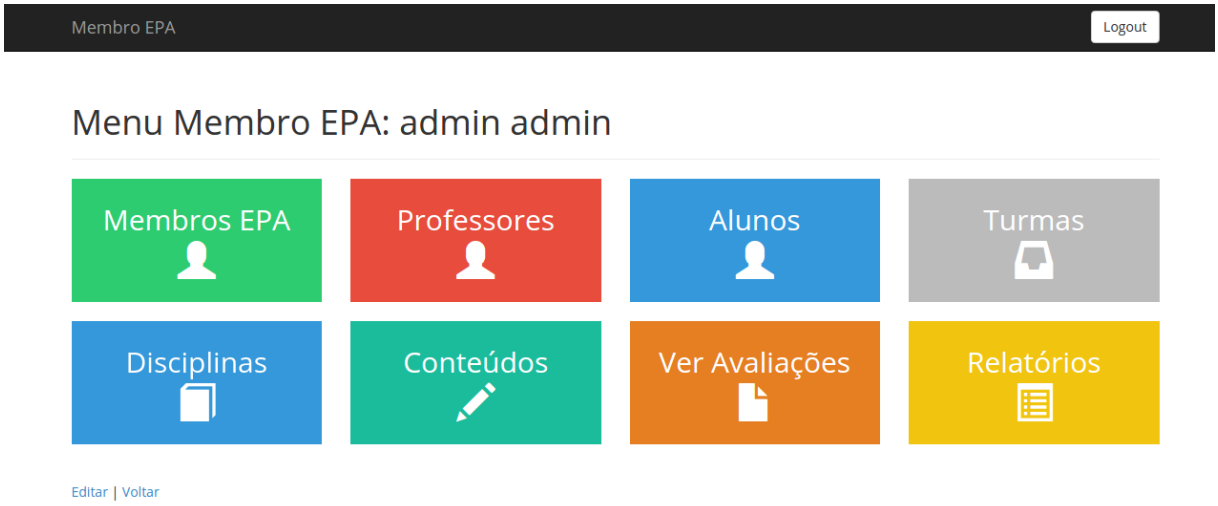


Figura 28 – Tela de menu de membro da EPA.

Uma das principais características do membro EPA é a possibilidade de realizar o gerenciamento de informações de suma importância para o funcionamento do sistema. Dentre elas, podemos citar as telas de gerenciamento de Professores (figura 29), gerenciamento de Turmas (figura 30) e gerenciamento de Conteúdos (figura 31). Nestas telas é possível realizar quatro tipos de ação: mostrar, editar, deletar e cadastrar novas informações (CRUD).

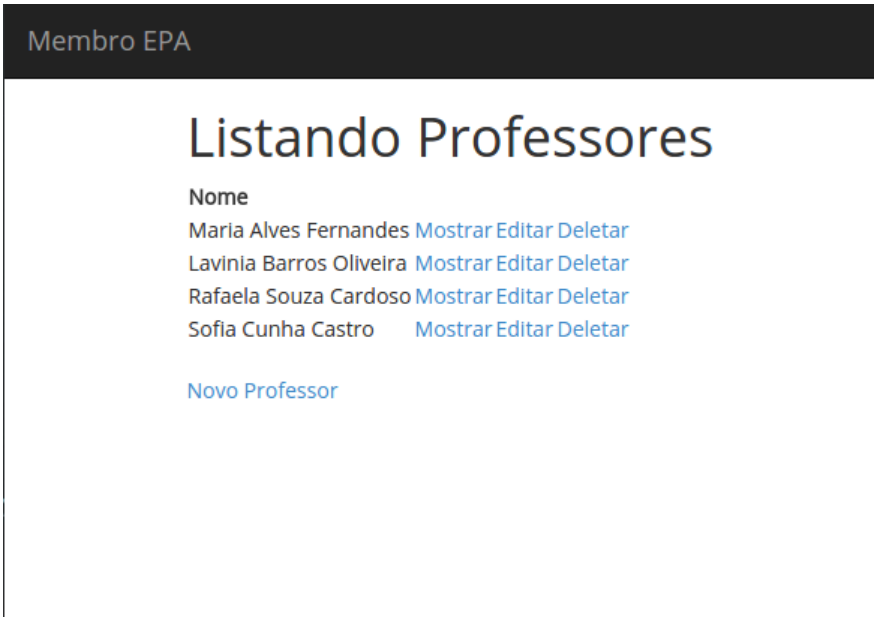


Figura 29 –Tela de gerenciamento de professores.

Figura 30 – Tela de gerenciamento de turmas.

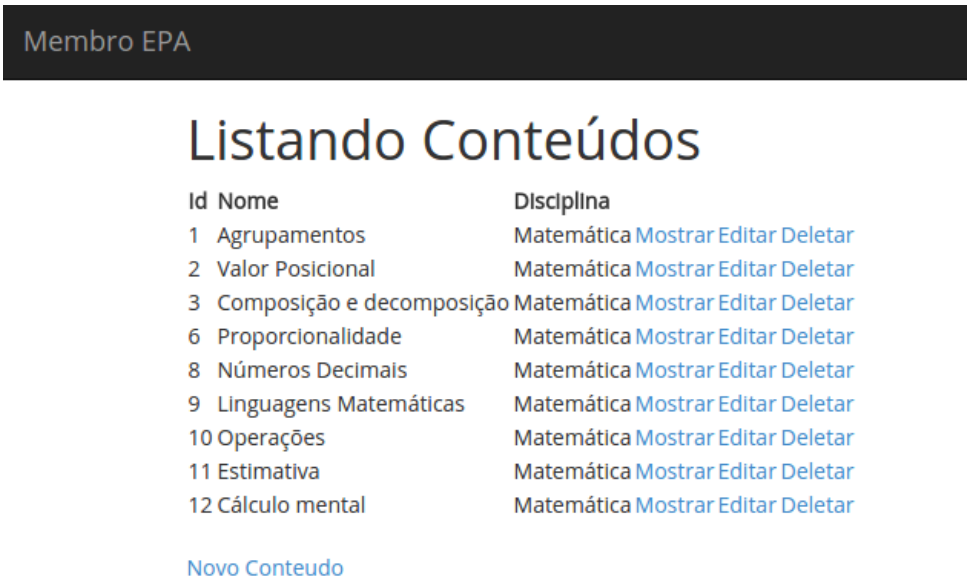


Figura 31 – Tela de gerenciamento de conteúdos.

Uma das opções nas telas de gerenciamento é acessar uma informação específica através da opção “Mostrar” e com isso conseguir realizar outras ações. Por exemplo, na tela de gerenciamento de turmas, ao selecionar a opção “Mostrar” em uma turma específica, o usuário

será redirecionado para uma tela que lhe permitirá cadastrar novas disciplinas na turma, verificar as disciplinas que já estão cadastradas na turma e adicionar um professor que irá lecionar a disciplina na turma em questão. Outra funcionalidade disponível é a inclusão de alunos na turma, com a possibilidade de selecionar vários alunos da lista simultaneamente, o que torna o processo mais rápido. Essas opções podem ser evidenciadas na figura 32.

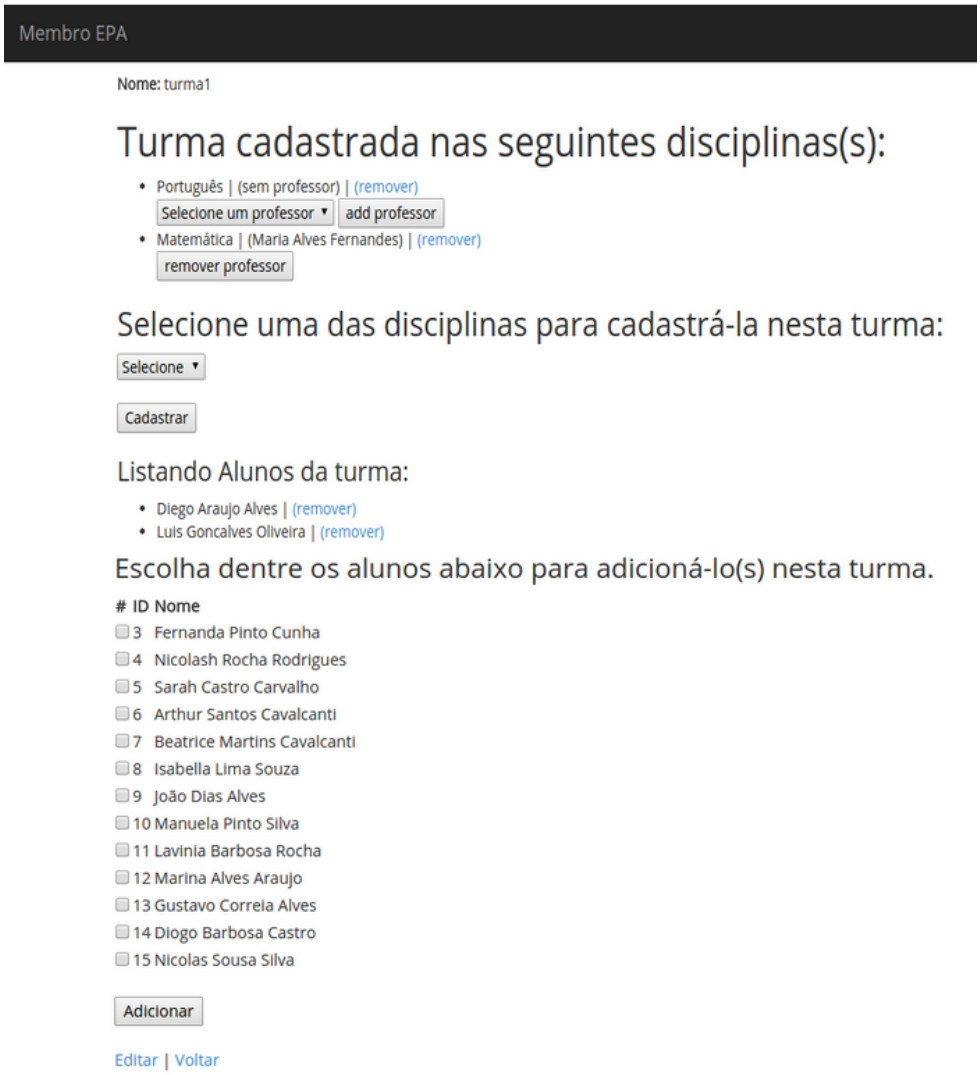


Figura 32 – Tela de gerenciar turma.

Na figura 33 evidenciamos as funcionalidades do campo “mostrar” no que se refere ao gerenciamento de disciplinas. O usuário poderá cadastrar novos conteúdos, incluí-los em turmas específicas e ainda, relacionar turmas às disciplinas.

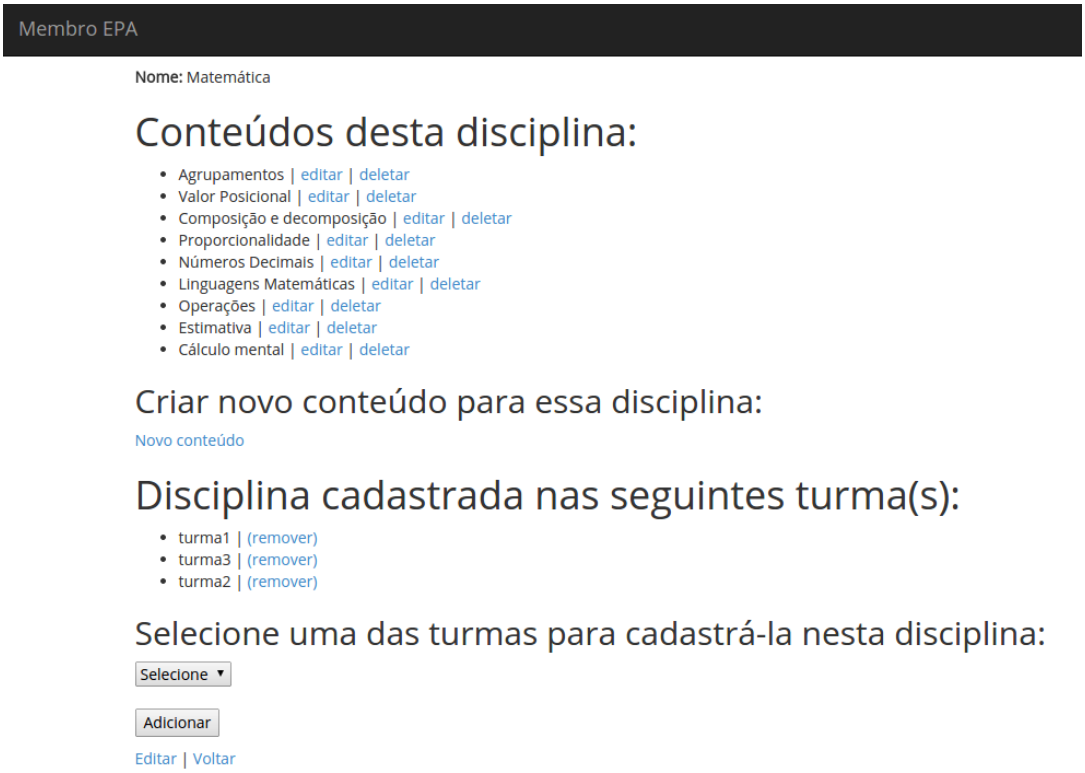


Figura 33 – Tela de gerenciar disciplina.

O menu "ver turmas", como pode ser visto na figura 34, oferece aos professores acesso rápido às suas turmas; é possível consultar a lista de turmas, bem como seus respectivos alunos.

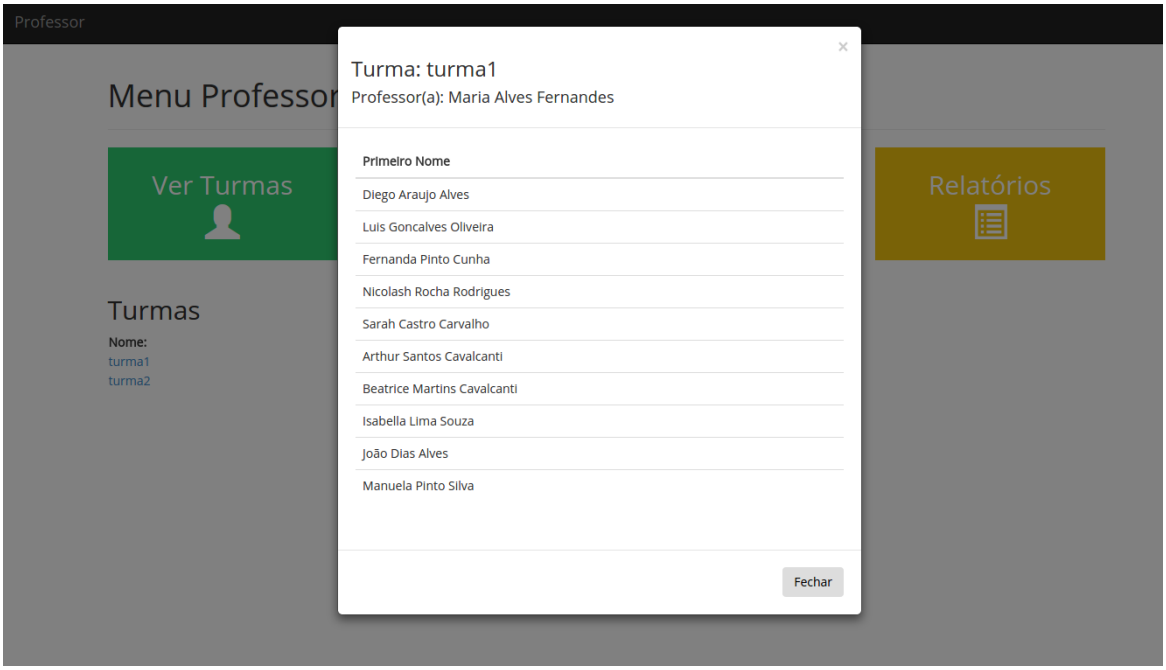


Figura 34 – Janela Ver Turmas ao clicar em "turma1".

Ao clicar na opção “Avaliar Alunos”, o professor será redirecionado para uma tela que irá mostrar todas as turmas em que ele esteja cadastrado. Ao clicar na turma escolhida para realizar a avaliação, será apresentado ao professor todas as disciplinas que ele esteja lecionando na turma. Clicando na disciplina, serão apresentados todos os conteúdos que ele poderá avaliar, como pode ser visto na figura 35.

Clicando no botão “Avaliar”, o professor será redirecionado para a página de avaliação; nesta página (figura 36), ele conseguirá avaliar os alunos de determinada turma para o conteúdo da disciplina selecionada.

Figura 35 – Tela para avaliar alunos.

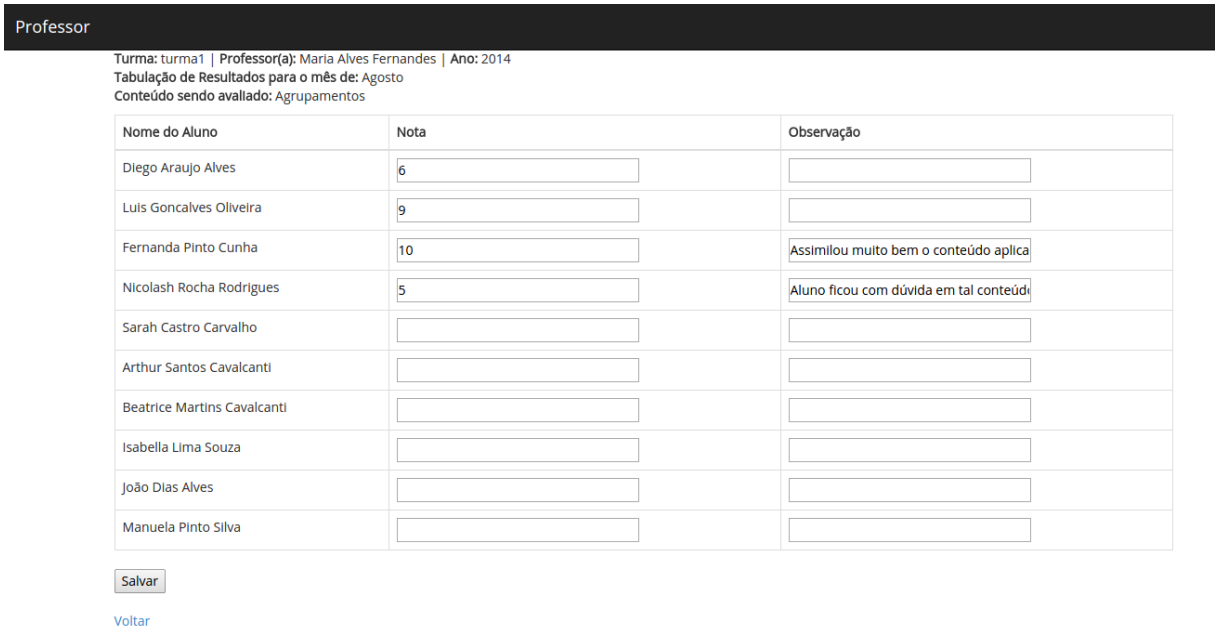


Figura 36 – Tela de avaliação de alunos.

6 CONSIDERAÇÕES FINAIS

Os objetivos do trabalho foram alcançados com sucesso, pois adquiriu-se conhecimento pleno, tanto teórico e prático, em desenvolvimento de software com técnica baseada em comportamento, na linguagem *Ruby* e no *framework Ruby on Rails*. Além disso, pode-se analisar e determinar os comportamentos do sistema que atende as necessidades desejadas pelo cliente (EPA) e construir uma versão de teste de um sistema para avaliação de desempenho de estudantes do Ensino Fundamental em Curitiba.

O maior desafio neste projeto foi a elaboração de bons requisitos (na forma de cenários) e a escrita desses requisitos utilizando a formatação correta que pudesse ser interpretada devidamente pela ferramenta de BDD. Foi necessário adaptar as estórias de usuário diversas vezes até que fizessem sentido e fossem aceitáveis em relação aos requisitos do sistema definidos pela EPA. Além disso, todo o código, sem exceção alguma, foi escrito apenas após um comportamento ser compilado, sendo constatado pelo *Cucumber* que a seção de código é essencial para o correto comportamento do sistema de acordo com o requisito relativo ao comportamento em questão. Na opinião dos autores deste TCC, este processo é motivador para o programador, pois ele se sente sob controle das funcionalidades reais do sistema sem ao menos vê-lo executando em um ambiente de produção.

A satisfação da EPA e dos professores, que puderam observar o *software* realizando as funções descritas por elas mesmas no início do projeto, foi melhor do que esperado, pois houve ganho de eficiência no tempo da realização das avaliações e o atendimento aos requisitos foi completo porque elas mesmas haviam auxiliado na especificação. No entanto, para que o *software* possa ser usado efetivamente, é necessário que seja futuramente implantado em um ambiente de produção com acesso na escola e que os dados de cada professor, aluno, conteúdo e etc. sejam mantidos, no mínimo, com frequência anual. Infelizmente, não houve tempo hábil para tanto durante a realização deste TCC.

O sucesso na realização deste projeto somente foi possível com a utilização de conhecimentos adquiridos no decorrer de todo o curso de Bacharelado em Sistemas de Informação, focado nas disciplinas de Análise e Projeto de Sistemas (na qual são ensinados conceitos de documentação e diagramas de sistemas), Engenharia de Software (na qual são ensinados os conceitos e processos da engenharia de software), Design de Interação (na qual são ensinados os conceitos de interface com usuários), Computação e Sociedade (na qual são discutidos os impactos da computação na sociedade) e Gerenciamento de Projetos (na qual é ensinado como gerenciar os diferentes recursos de um projeto).

Enfim, este trabalho pode vir a ser útil a outros estudantes que desenvolvam projetos com BDD, *Ruby* e *Ruby on Rails*, pois, além de explicar os procedimentos que podem ser seguidos, mostra um exemplo concreto do desenvolvimento de um *software* que soluciona um problema do mundo real. Além disso, este projeto de TCC pode vir a ser útil a estudantes dispostos a trabalhar no aprimoramento e/ou implantação do *software* desenvolvido em outros ambientes.

REFERÊNCIAS BIBLIOGRÁFICAS

BRASIL. Parâmetros Curriculares Nacionais para o Ensino Fundamental. Brasil: CNE/ CEB, 1997. Disponível em: <http://portaldoprofessor.mec.gov.br/linksCursosMateriais.html?categoria=23>. Acessado em: 09/03/2013.

RUBY, **A Programmer's Best Friend**. Disponível em: <http://www.ruby-lang.org/pt/sobre-o-ruby>. Acesso em: 15/02/2013.

AKITA, Fabio. **Repensando a web com Rails**. Rio de Janeiro: Brasport, 2006.

NORTH, D. **Introducing Behaviour-Driven Development**. 2006. Disponível em: <http://dannorth.net/introducing-bdd>. Acesso em: 23/02/2013.

GAMMA, Erich; JOHNSON, Ralph; HELM, Richard; VLISSIDES, John. **Padrões de projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2002 364 p.

BECK, K. Test-Driven Development By Example. Addison Wesley, 2002.

WYNNE, Mat; HELLESOY, Aslak. The Cucumber Book: Behaviour-Driven Development for Testers and Developers. EUA: Jacquelyn Carter, 2011.

FAYAD, M.E.; SCHMIDT, D.C.;JOHNSON, R.E. Object-Oriented Application Frameworks: Implementation and Experience. Wiley, NY, 1997.

HEROKU, **A Cloud Application Platform**. Disponível em: <http://about.heroku.com/>. Acesso em: 12/01/2013.

PACHECO, Diego. **PaaS, Cloud Computing, Virtualização e o Futuro**. 2009. Disponível em: <http://imasters.com.br/artigo/14165/gerenciadeprojetos/paas_cloud_computing_virtualizacao_e_o_futuro_parte_01/>. Acesso em 12/01/2013.

THOMAS, D. H. **Desenvolvimento Web Ágil com Rails**. Porto Alegre: Bookman, 2008.

FILHO, Luiz C. M. TDD On Rails – Desenvolvimento Guiado a Testes em Aplicações Web com Framework Rails.,2012.

JANZEN, D. Software Architecture Improvement through Test-Driven Development., 2005.

TORRES, Joaquim. MVC e Ruby on Rails, uma visão simplificada., 2008.

CAELUM. **RR-71Desenvolvimento Ágil para Web 2.0 com Ruby on Rails**, 2013. Disponivel em: <http://www.caelum.com.br/apostilas/> Acesso em: 11 abr. 13

MONTEIRO, J. Ruby on Rails : Ruby on Rails Brasil. **Ruby on Rails Brasil**, 2012.Disponivel em: < http://rubyonrails.com.br/ >. Acesso em: 11 abr. 13

CHACON, Scott. **Pro Git: Community Book**. Disponível em: <http://book.git-scm.com/index.html>. Acesso em: 12/01/2013.

GIT. **Git Documentation**. <http://git-scm.com/documentation>. Acesso em 25/02/2013.

GITHUB. **Features / Project Management: GitHub.** Disponível em: <https://github.com/features/projects>. Acesso em 09/03/2013.

D'OLIVEIRA, Ruben. **Code Editor Review: Sublime Text.** Disponível em: <http://www.1stwebdesigner.com/design/sublime-text-code-editor-review/>. Acesso em 28/02/2013.

BRUCE, James.**Try out Sublime Text 2 for your cross-platform code editing needs.** Disponível em: <http://www.makeuseof.com/tag/sublime-text-2-crossplatform-code-editing/>. Acesso em 07/03/2013.

BOOTSTRAP. **Bootstrap v2.3.0 - BootstrapDocs.** Disponível em: <http://bootstrapdocs.com/v2.3.0/docs/index.html>. Acesso em 11/03/2013.

URUBATAN, Rodrigo. Ruby on Rails - Desenvolvimento Fácil e Rápido de Aplicações Web, Novatec, 2009.

Câmara de Ensino Fundamental de Curitiba. **Parecer N.º 487/99:** Projeto de Implantação dos Ciclos de Aprendizagem (1ª. a 8.ª séries) na Rede Municipal de Ensino de Curitiba. 1999. 5 p. Disponível em: http://www.cidadedoconhecimento.org.br/cidadedoconhecimento/legislacao/arquivos/legislacao_8.htm Acessado em 2 de abril de 2013.

Conselho Municipal de Educação de Curitiba. **Deliberação N.º 01/2008:** organização das matrículas para o ingresso no Ensino Fundamental obrigatório de 9 (nove) anos de duração. 2008. Disponível em: http://www.cidadedoconhecimento.org.br/cidadedoconhecimento/legislacao/arquivos/legislacao_152.pdf Acessado em 2 de abril de 2013.

Secretaria Municipal de Educação de Curitiba. **Portaria N.º 26/2005:** fixa o número de educandos, para efeito de composição de turmas, das unidades de Educação e Ensino. 2005. Disponível em: http://www.cidadedoconhecimento.org.br/cidadedoconhecimento/legislacao/arquivos/legislacao_111.pdf Acessado em 2 de abril de 2013.

PARANÁ. **Deliberação N.º 007/1999:** Normas Gerais para Avaliação do Aproveitamento Escolar, Recuperação de
Estudos e Promoção de Alunos, do Sistema Estadual de Ensino, em Nível do
Ensino Fundamental e Médio. 1999. Disponível em: http://celepar7cta.pr.gov.br/seed/deliberacoes.nsf/7b2a997ca37239c3032569ed005fb978/b15be00846f01f20032569f1004972fb/$FILE/_88himoqb2clp631u6dsg30dpd64sjie8_.pdf
Acessado em 2 de abril de 2013.

BRASIL. **Lei N.º 11.114, de 16 de maio de 2005:** Altera os arts. 6o, 30, 32 e 87 da Lei no 9.394, de 20 de dezembro de 1996, com o objetivo de tornar obrigatório o início do ensino fundamental aos seis anos de idade. 2005. Disponível em:

http://www.planalto.gov.br/ccivil_03/_Ato2004-2006/2005/Lei/L11114.htm Acessado em 2 abril de 2013.

BRASIL. **Lei nº 9.394 de 20 de dezembro de 1996.** Estabelece as diretrizes e bases da educação nacional. Disponível em: http://www.planalto.gov.br/ccivil_03/Leis/L9394.htm Acessado em 20 de abril de 2013.

BRASIL. **Plano Nacional de Educação.** Brasil: MEC/SEB, 2001. Disponível em: http://portal.mec.gov.br/arquivos/pdf/pne.pdf Acessado em 3 de abril de 2013.

BRASIL. **Diretrizes Curriculares Nacionais para o Ensino Fundamental.** Brasil: CNE/CEB, 1998. Disponível em: http://portal.mec.gov.br/arquivos/pdf/rceb02_98.pdf Acessado em 20 de abril de 2013.

APÊNDICE

APÊNDICE A – FUNCIONALIDADES

A seguir, a impressão completa do arquivo de funcionalidades do projeto.

```
# language: pt
Funcionalidade: Gerenciar Alunos
  Como um membro do EPA
  Eu quero poder cadastrar, editar, listar e apagar alunos
  Para que eu possa relacionar os alunos com as turmas

  Cenário: Listando todas os alunos                    # features/gerenciar_alunos.feature:8
    Dado que existam os alunos "Estevan", "Ricardo", "Thaise" # features/step_definitions/gerenciar_alunos.rb:3
    E que eu esteja na página de administração de alunos # features/step_definitions/gerenciar_alunos.rb:9
    Então eu devo ver "Estevan"                        # features/step_definitions/gerenciar_disciplinas.rb:13
    E eu devo ver "Ricardo"                            # features/step_definitions/gerenciar_disciplinas.rb:13
    E eu devo ver "Thaise"                             # features/step_definitions/gerenciar_disciplinas.rb:13

  Cenário: Criar um novo aluno                         # features/gerenciar_alunos.feature:15
    Dado que existam os alunos "Estevan", "Ricardo", "Thaise" # features/step_definitions/gerenciar_alunos.rb:3
    E que eu esteja na página de administração de alunos # features/step_definitions/gerenciar_alunos.rb:9
    Quando eu clico em "Novo Aluno"                    # features/step_definitions/gerenciar_disciplinas.rb:17
    E eu preencho o campo "Nome" com "Sâmia"           # features/step_definitions/gerenciar_disciplinas.rb:21
```

E eu pressiono o botão "Salvar" # features/step_definitions/gerenciar_disciplinas.rb:25
Então a seguinte mensagem deve ser apresentada: "Aluno criado com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29
E eu devo ver "Estevan" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Ricardo" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Thaise" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Sâmia" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Editar um aluno # features/gerenciar_alunos.feature:27
Dado que existam os alunos "Estevan", "Ricardo", "Thaise" # features/step_definitions/gerenciar_alunos.rb:3
E que eu esteja na página de administração de alunos # features/step_definitions/gerenciar_alunos.rb:9
Quando eu clico em "Editar" ao lado de "Estevan" # features/step_definitions/gerenciar_disciplinas.rb:33
E eu preencho o campo "Nome" com "Estevan Jantsk" # features/step_definitions/gerenciar_disciplinas.rb:21
E eu pressiono o botão "Salvar" # features/step_definitions/gerenciar_disciplinas.rb:25
Então a seguinte mensagem deve ser apresentada: "Aluno atualizado com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29

Cenário: Deletar um aluno # features/gerenciar_alunos.feature:35
Dado que existam os alunos "Estevan", "Ricardo", "Thaise" # features/step_definitions/gerenciar_alunos.rb:3
E que eu esteja na página de administração de alunos # features/step_definitions/gerenciar_alunos.rb:9
Quando eu clico em "Deletar" ao lado de "Estevan" # features/step_definitions/gerenciar_disciplinas.rb:33

Então eu NÃO devo ver "Estevan" # features/step_definitions/gerenciar_disciplinas.rb:39
E eu devo ver "Ricardo" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Thaise" # features/step_definitions/gerenciar_disciplinas.rb:13

language: pt
Funcionalidade: Gerenciar Conteudos
Como um membro do EPA
Eu quero poder cadastrar, editar, listar e apagar conteúdos
Para que eu possa adicionar os conteudos nas disciplinas

Cenário: Listando todas os conteudos # features/gerenciar_conteudos.feature:8
Dado que existam os conteudos "Agrupamentos", "Valor posicional", "Números decimais" pertencentes a disciplina "Matemática" # features/step_definitions/gerenciar_conteudos.rb:3
E que eu esteja na página de administração de conteúdos # features/step_definitions/gerenciar_conteudos.rb:12
Então eu devo ver "Agrupamentos" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Valor posicional" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Números decimais" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Criar um novo conteudo # features/gerenciar_conteudos.feature:15
Dado que existam os conteudos "Agrupamentos", "Valor posicional", "Números decimais" pertencentes a disciplina "Matemática" # features/step_definitions/gerenciar_conteudos.rb:3
E que eu esteja na página de administração de conteúdos # features/step_definitions/gerenciar_conteudos.rb:12
Quando eu clico em "Novo Conteudo" # features/step_definitions/gerenciar_disciplinas.rb:17

E eu preencho o campo "Nome" com "Proporcionalidade" # features/step_definitions/gerenciar_disciplinas.rb:21

E eu seleciono a disciplina "Matemática" no campo "Disciplina" # features/step_definitions/gerenciar_conteudos.rb:16

E eu pressiono o botão "Salvar" # features/step_definitions/gerenciar_disciplinas.rb:25

Então a seguinte mensagem deve ser apresentada: "Conteudo criado com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29

E eu devo ver "Agrupamentos" # features/step_definitions/gerenciar_disciplinas.rb:13

E eu devo ver "Valor posicional" # features/step_definitions/gerenciar_disciplinas.rb:13

E eu devo ver "Números decimais" # features/step_definitions/gerenciar_disciplinas.rb:13

E eu devo ver "Proporcionalidade" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Editar um conteudo # features/gerenciar_conteudos.feature:28

Dado que existam os conteudos "Agrupamentos", "Valor posicional", "Números decimais" pertencentes a disciplina "Matemática" # features/step_definitions/gerenciar_conteudos.rb:3

E que eu esteja na página de administração de conteúdos # features/step_definitions/gerenciar_conteudos.rb:12

Quando eu clico em "Editar" ao lado de "Agrupamentos" # features/step_definitions/gerenciar_disciplinas.rb:33

E eu preencho o campo "Nome" com "Agrup" # features/step_definitions/gerenciar_disciplinas.rb:21

E eu pressiono o botão "Salvar" # features/step_definitions/gerenciar_disciplinas.rb:25

Então a seguinte mensagem deve ser apresentada: "Conteudo atualizado com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29

Cenário: Deletar um conteudo # features/gerenciar_conteudos.feature:36

Dado que existam os conteudos "Agrupamentos", "Valor posicional", "Números decimais" pertencentes a disciplina "Matemática" # features/step_definitions/gerenciar_conteudos.rb:3
E que eu esteja na página de administração de conteúdos # features/step_definitions/gerenciar_conteudos.rb:12
Quando eu clico em "Deletar" ao lado de "Agrupamentos" # features/step_definitions/gerenciar_disciplinas.rb:33
Então eu NÃO devo ver "Agrupamentos" # features/step_definitions/gerenciar_disciplinas.rb:39
E eu devo ver "Valor posicional" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Números decimais" # features/step_definitions/gerenciar_disciplinas.rb:13

language: pt
Funcionalidade: Gerenciar Disciplinas
Como um membro do EPA
Eu quero poder cadastrar, editar, listar e apagar disciplinas
Para que eu possa relacionar as disciplinas com conteúdos e turmas

Cenário: Listando todas as disciplinas # features/gerenciar_disciplinas.feature:8
Dado que existam as disciplinas "Matemática", "Português" e "Ciências" # features/step_definitions/gerenciar_disciplinas.rb:3
E que eu esteja na página de administração de disciplinas # features/step_definitions/gerenciar_disciplinas.rb:9
Então eu devo ver "Matemática" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Português" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Ciências" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Criar uma nova disciplina # features/gerenciar_disciplinas.feature:15

Dado que existam as disciplinas "Matemática", "Português" e "Ciências" # features/step_definitions/gerenciar_disciplinas.rb:3
E que eu esteja na página de administração de disciplinas # features/step_definitions/gerenciar_disciplinas.rb:9
Quando eu clico em "Nova Disciplina" # features/step_definitions/gerenciar_disciplinas.rb:17
E eu preencho o campo "Nome" com "Geografia" # features/step_definitions/gerenciar_disciplinas.rb:21
E eu pressiono o botão "Salvar" # features/step_definitions/gerenciar_disciplinas.rb:25
Então a seguinte mensagem deve ser apresentada: "Disciplina criada com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29
E eu devo ver "Matemática" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Português" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Ciências" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Geografia" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Editar uma disciplina # features/gerenciar_disciplinas.feature:27
Dado que existam as disciplinas "Matemática", "Português" e "Ciências" # features/step_definitions/gerenciar_disciplinas.rb:3
E que eu esteja na página de administração de disciplinas # features/step_definitions/gerenciar_disciplinas.rb:9
Quando eu clico em "Editar" ao lado de "Matemática" # features/step_definitions/gerenciar_disciplinas.rb:33
E eu preencho o campo "Nome" com "Mat" # features/step_definitions/gerenciar_disciplinas.rb:21
E eu pressiono o botão "Salvar" # features/step_definitions/gerenciar_disciplinas.rb:25

Então a seguinte mensagem deve ser apresentada: "Disciplina atualizada com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29

Cenário: Deletar uma disciplina # features/gerenciar_disciplinas.feature:35
Dado que existam as disciplinas "Matemática", "Português" e "Ciências" # features/step_definitions/gerenciar_disciplinas.rb:3
E que eu esteja na página de administração de disciplinas # features/step_definitions/gerenciar_disciplinas.rb:9
Quando eu clico em "Deletar" ao lado de "Matemática" # features/step_definitions/gerenciar_disciplinas.rb:33
Então eu NÃO devo ver "Matemática" # features/step_definitions/gerenciar_disciplinas.rb:39
E eu devo ver "Português" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Ciências" # features/step_definitions/gerenciar_disciplinas.rb:13

language: pt
Funcionalidade: Gerenciar Professores
Como um membro do EPA
Eu quero poder cadastrar, editar, listar e apagar professores
Para que eu possa relacionar os professores com as turmas

Cenário: Listando todas os professores # features/gerenciar_professores.feature:8
Dado que existam os professores "Gilda", "Maria", "Claudia" # features/step_definitions/gerenciar_professores.rb:3
E que eu esteja na página de administração de professores # features/step_definitions/gerenciar_professores.rb:9
Então eu devo ver "Gilda" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Maria" # features/step_definitions/gerenciar_disciplinas.rb:13

E eu devo ver "Claudia" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Criar um novo professor # features/gerenciar_professores.feature:15
Dado que existam os professores "Gilda", "Maria", "Claudia" # features/step_definitions/gerenciar_professores.rb:3
E que eu esteja na página de administração de professores # features/step_definitions/gerenciar_professores.rb:9
Quando eu clico em "Novo Professor" # features/step_definitions/gerenciar_disciplinas.rb:17
E eu preencho o campo "Nome" com "Stadzisz" # features/step_definitions/gerenciar_disciplinas.rb:21
E eu pressiono o botão "Criar Professor" # features/step_definitions/gerenciar_disciplinas.rb:25
Então a seguinte mensagem deve ser apresentada: "Professor criado com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29
E eu devo ver "Gilda" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Maria" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Claudia" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Stadzisz" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Editar um professor # features/gerenciar_professores.feature:27
Dado que existam os professores "Gilda", "Maria", "Claudia" # features/step_definitions/gerenciar_professores.rb:3
E que eu esteja na página de administração de professores # features/step_definitions/gerenciar_professores.rb:9
Quando eu clico em "Editar" ao lado de "Maria" # features/step_definitions/gerenciar_disciplinas.rb:33

E eu preencho o campo "Nome" com "Mariangela" # features/step_definitions/gerenciar_disciplinas.rb:21
E eu pressiono o botão "Atualizar Professor" # features/step_definitions/gerenciar_disciplinas.rb:25
Então a seguinte mensagem deve ser apresentada: "Professor atualizado com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29

Cenário: Deletar um professor # features/gerenciar_professores.feature:35
Dado que existam os professores "Gilda", "Maria", "Claudia" # features/step_definitions/gerenciar_professores.rb:3
E que eu esteja na página de administração de professores # features/step_definitions/gerenciar_professores.rb:9
Quando eu clico em "Deletar" ao lado de "Gilda" # features/step_definitions/gerenciar_disciplinas.rb:33
Então eu NÃO devo ver "Gilda" # features/step_definitions/gerenciar_disciplinas.rb:39
E eu devo ver "Maria" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "Claudia" # features/step_definitions/gerenciar_disciplinas.rb:13

language: pt
Funcionalidade: Gerenciar Turmas
Como um membro do EPA
Eu quero poder cadastrar, editar, listar e apagar turmas
Para que eu possa relacionar as turmas com professores, alunos e disciplinas

Cenário: Listando todas as turmas # features/gerenciar_turmas.feature:8
Dado que existam as turmas "4MA2013", "2TB2013" # features/step_definitions/gerenciar_turmas.rb:3
E que eu esteja na página de administração de turmas # features/step_definitions/gerenciar_turmas.rb:8
Então eu devo ver "4MA2013" # features/step_definitions/gerenciar_disciplinas.rb:13

E eu devo ver "2TB2013" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Criar uma nova turma # features/gerenciar_turmas.feature:14
Dado que existam as turmas "4MA2013", "2TB2013" # features/step_definitions/gerenciar_turmas.rb:3
E que eu esteja na página de administração de turmas # features/step_definitions/gerenciar_turmas.rb:8
Quando eu clico em "Nova Turma" # features/step_definitions/gerenciar_disciplinas.rb:17
E eu preencho o campo "Nome" com "4MB2013" # features/step_definitions/gerenciar_disciplinas.rb:21
E eu pressiono o botão "Salvar" # features/step_definitions/gerenciar_disciplinas.rb:25
Então a seguinte mensagem deve ser apresentada: "Turma criada com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29
E eu devo ver "4MA2013" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "2TB2013" # features/step_definitions/gerenciar_disciplinas.rb:13
E eu devo ver "4MB2013" # features/step_definitions/gerenciar_disciplinas.rb:13

Cenário: Editar uma turma # features/gerenciar_turmas.feature:25
Dado que existam as turmas "4MA2013", "2TB2013" # features/step_definitions/gerenciar_turmas.rb:3
E que eu esteja na página de administração de turmas # features/step_definitions/gerenciar_turmas.rb:8
Quando eu clico em "Editar" ao lado de "4MA2013" # features/step_definitions/gerenciar_disciplinas.rb:33
E eu preencho o campo "Nome" com "4TA2013" # features/step_definitions/gerenciar_disciplinas.rb:21
E eu pressiono o botão "Salvar" # features/step_definitions/gerenciar_disciplinas.rb:25

Então a seguinte mensagem deve ser apresentada: "Turma atualizada com sucesso." # features/step_definitions/gerenciar_disciplinas.rb:29

Cenário: Deletar uma turma # features/gerenciar_turmas.feature:33
Dado que existam as turmas "4MA2013", "2TB2013" # features/step_definitions/gerenciar_turmas.rb:3
E que eu esteja na página de administração de turmas # features/step_definitions/gerenciar_turmas.rb:8
Quando eu clico em "Deletar" ao lado de "4MA2013" # features/step_definitions/gerenciar_disciplinas.rb:33
Então eu NÃO devo ver "4MA2013" # features/step_definitions/gerenciar_disciplinas.rb:39
E eu devo ver "2TB2013" # features/step_definitions/gerenciar_disciplinas.rb:13

language: pt
Funcionalidade: Realizar Avaliacao
Como um professor interessado nas avaliações
Com o objetivo de avaliar os alunos
Os professores devem poder visualizar, realizar e editar as avaliações dos alunos de suas respectivas turmas

Cenário: Acessar a página do professor # features/realizar_avalicao.feature:8
Dado que exista a professora "Fabiane" # features/step_definitions/realizar_avaliacao.rb:3
E que a professora "Fabiane" esteja na página Professor # features/step_definitions/realizar_avaliacao.rb:7
Então devem aparecer os links "Ver Turmas", "Avaliar Alunos", "Ver Avaliações", "Relatórios" # features/step_definitions/realizar_avaliacao.rb:13

@javascript
Cenário: Ver Turmas # features/realizar_avalicao.feature:14

Dado que exista a professora "Fabiane" # features/step_definitions/realizar_avaliacao.rb:3
E que a professora "Fabiane" esteja na página Professor # features/step_definitions/realizar_avaliacao.rb:7
E que a disciplina "Matemática" esteja cadastrada na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:20
E que a professora "Fabiane" dê aula de "Matemática" na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:27
Quando a professora clicar em "Ver Turmas" # features/step_definitions/realizar_avaliacao.rb:32
Então liste as turmas que a professora dê aula # features/step_definitions/realizar_avaliacao.rb:36

@javascript
Cenário: Visualizar alunos de turma # features/realizar_avalicao.feature:23
Dado que exista a professora "Fabiane" # features/step_definitions/realizar_avaliacao.rb:3
E que a professora "Fabiane" esteja na página Professor # features/step_definitions/realizar_avaliacao.rb:7
E que a disciplina "Matemática" esteja cadastrada na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:20
E que a professora "Fabiane" dê aula de "Matemática" na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:27
E que os alunos "Joaozinho", "Pedrinho", "Carlota" perteçam a turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:51
Quando a professora clicar em "Ver Turmas" # features/step_definitions/realizar_avaliacao.rb:32
E a professora clicar na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:62
E liste os alunos que pertencem a turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:66

Cenário: Ver Disciplinas # features/realizar_avalicao.feature:33

Dado que exista a professora "Fabiane" # features/step_definitions/realizar_avaliacao.rb:3
E que a professora "Fabiane" esteja na página Professor # features/step_definitions/realizar_avaliacao.rb:7
E que a disciplina "Matemática" esteja cadastrada na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:20
E que a professora "Fabiane" dê aula de "Matemática" na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:27
E que os alunos "Joaozinho", "Pedrinho", "Carlota" perteçam a turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:51
Quando a professora clicar em "Avaliar Alunos" # features/step_definitions/realizar_avaliacao.rb:32
Então liste as turmas da professora com suas respectivas disciplinas # features/step_definitions/realizar_avaliacao.rb:86

@javascript
Cenário: Ver conteúdos de disciplina # features/realizar_avalicao.feature:43
Dado que exista a professora "Fabiane" # features/step_definitions/realizar_avaliacao.rb:3
E que a professora "Fabiane" esteja na página Professor # features/step_definitions/realizar_avaliacao.rb:7
E que a disciplina "Matemática" esteja cadastrada na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:20
E que a disciplina "Matemática" possua os conteúdos "Agrupamentos", "Valor posicional" # features/step_definitions/realizar_avaliacao.rb:104
E que a professora "Fabiane" dê aula de "Matemática" na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:27
E que os alunos "Joaozinho", "Pedrinho", "Carlota" perteçam a turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:51
Quando a professora clicar em "Avaliar Alunos" # features/step_definitions/realizar_avaliacao.rb:32
E a professora clicar em "Matemática" # features/step_definitions/realizar_avaliacao.rb:32

Então a página deve listar os conteúdos "Agrupamentos", "Valor posicional" # features/step_definitions/realizar_avaliacao.rb:114

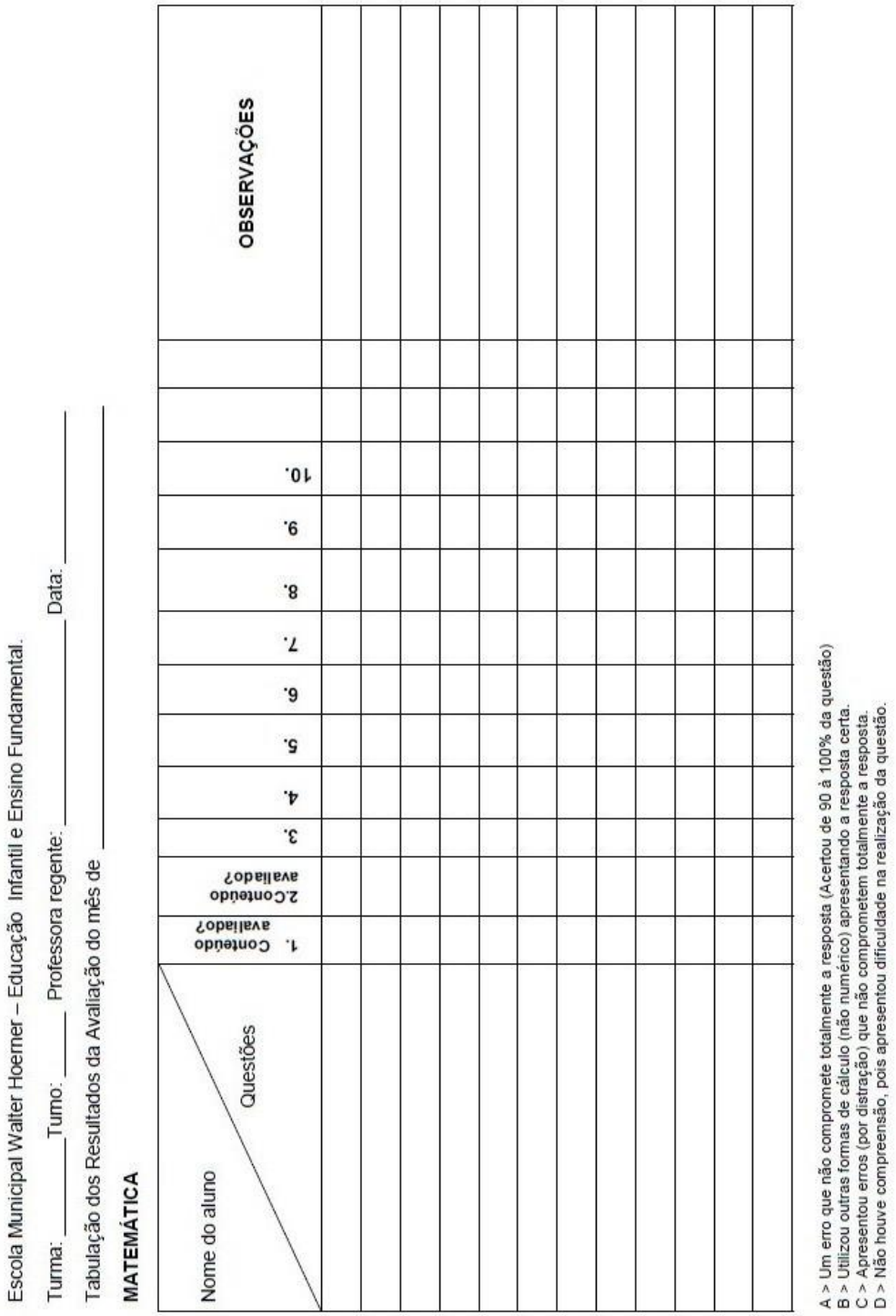
@javascript
Cenário: Avaliar conteúdo # features/realizar_avalicao.feature:55
Dado que exista a professora "Fabiane" # features/step_definitions/realizar_avaliacao.rb:3
E que a professora "Fabiane" esteja na página Professor # features/step_definitions/realizar_avaliacao.rb:7
E que a disciplina "Matemática" esteja cadastrada na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:20
E que a disciplina "Matemática" possua os conteúdos "Agrupamentos", "Valor posicional" # features/step_definitions/realizar_avaliacao.rb:104
E que a professora "Fabiane" dê aula de "Matemática" na turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:27
E que os alunos "Joaozinho", "Pedrinho", "Carlota" perteçam a turma "4MA2013" # features/step_definitions/realizar_avaliacao.rb:51
Quando a professora clicar em "Avaliar Alunos" # features/step_definitions/realizar_avaliacao.rb:32
E a professora clicar em "Matemática" # features/step_definitions/realizar_avaliacao.rb:32
E a professora escolher avaliar o conteudo "Agrupamentos" no mês atual # features/step_definitions/realizar_avaliacao.rb:132
E a professora conseguir atribuir notas aos alunos da sala # features/step_definitions/realizar_avaliacao.rb:149
E a professora desejar salvar # features/step_definitions/realizar_avaliacao.rb:165
Então a professora avaliou o conteudo no turma escolhida # features/step_definitions/realizar_avaliacao.rb:169

26 scenarios (26 passed)
178 steps (178 passed)
0m32.987s

ANEXOS

ANEXO A – Tabela de avaliação de matemática

Esta é a tabela feita no *Microsoft Word*, utilizada como *template* pelos funcionários das escolas para que os professores realizassem cada avaliação de matemática manualmente.



Escola Municipal Walter Hoerner – Educação Infantil e Ensino Fundamental.

Turma: _____ Turno: _____ Professora regente: _____ Data: _____

Tabulação dos Resultados da Avaliação do mês de _____

MATEMÁTICA

Nome do aluno / Questões	1. Conteúdo avaliado?	2.Conteúdo avaliado?	3.	4.	5.	6.	7.	8.	9.	10.			OBSERVAÇÕES

A > Um erro que não compromete totalmente a resposta (Acertou de 90 à 100% da questão)
B > Utilizou outras formas de cálculo (não numérico) apresentando a resposta certa.
C > Apresentou erros (por distração) que não comprometem totalmente a resposta.
D > Não houve compreensão, pois apresentou dificuldade na realização da questão.

ANEXO B – Conteúdos de matemática no ciclo II 2ª etapa

A seguir, uma tabela com os objetivos e conteúdos elaborados pelo MEC e, à direita, a recomendação feita pela EPA de qual trimestre cada conteúdo deve ser lecionado.

Objetivos	Conteúdos	1º	2º	3º
1. Compreender os princípios de organização do Sistema de Numeração Decimal (classe dos milhões) e valer-se deste para registrar, elaborar e resolver situações-problema em diferentes contextos.	- Agrupamentos .	X	X	X
	- Valor posicional.	X	X	X
	- Composição e decomposição.	X	X	X
	- Proporcionalidade (relação multiplicativa entre duas grandezas, dois números ou duas medidas, por exemplo, ao contarmos a quantidade de rodas que há em um estacionamento de carros, a quantidade de rodas aumenta conforme o número de carros).	X	X	X
	- Números decimais			
2. Utilizar-se da linguagem oral e da linguagem escrita para comunicar-se e produzir escritas matemáticas, na resolução de situações-problema de diferentes contextos.	- Linguagens matemáticas.	X	X	X
	- Operações.	X	X	X
	- Estimativa.	X	X	X
	- Cálculo mental.	X	X	X
	- Proporcionalidade.	X	X	X
	- Combinatória.	X	X	X
	- Probabilidade.	X	X	X
3. Analisar, coletar e representar informações que são apresentadas em linguagem gráfica, percebendo a intencionalidade com que elas foram representadas e a freqüência de acontecimentos	- Estatística: tabelas, gráfico de barras, colunas, setores, linhas e outros.	X	X	X
	- Probabilidade	X	X	X

previsíveis ou aleatórios, por meio de recursos estatísticos e probabilísticos.				
4. Fazer uso dos sistemas de medidas, comparando e estabelecendo relações entre as grandezas, assim como fazendo estimativas e probabilizando resultados.	- Medida de tempo: hora, minutos, segundos. - Medida de valor monetário: reais e centavos na composição das demais quantidades. - Medida de massa: quilograma e grama. - Medida de capacidade: litro, ml (Para compor 1 litro são necessários quantos copos de 200 ml ou 500 ml?). - Comprimento: km, m, cm, mm, cálculo do perímetro. - Medida de superfície: km2, m2, cálculo da área. Medida de volume: m3, cm3, cálculo do volume.		X X X X X X X	
5. Ampliar o Sistema de Numeração Decimal dos números naturais para os racionais, reconhecendo as relações entre as operações e suas diferentes representações.	- Representação fracionária. - Frações de unidade e de quantidade. - Equivalência de frações. - Representação decimal. - Operações com números decimais. - Porcentagem.	X X X X X X		
6. Orientar-se no espaço, interpretando e representando a localização e a movimentação de pessoas e objetos, a partir de pontos de referência, utilizando	- Noções topológicas: envolvem relações num mesmo objeto ou entre um objeto e outros elementos do espaço (aberto/fechado, interior/exterior, longe/perto, separado/unido,			X

corretamente a linguagem matemática.	contínuo/descontinuo, alto/baixo, vizinhança, fronteira). - Lateralidade: direita e esquerda. - Representação do espaço (mapas, malhas quadriculadas, maquetes e qualquer outro tipo de representação).			X X
7. Identificar características das figuras geométricas por meio de descrições orais, construções e representações, percebendo semelhanças e diferenças entre os objetos do espaço e do plano.	- Formas tridimensionais: poliedros e corpos redondos. - Formas bidimensionais: polígonos e círculos. - Noções projetivas: envolvem relações entre a figura e o sujeito, mantendo determinados elementos invariantes (noções de direita, esquerda, em cima, embaixo, na frente, atrás, etc.) numa projeção. - Noções euclidianas: investigam o que ocorre com as figuras geométricas quando estas sofrem deslocamentos, mantendo suas características (forma, dimensão). - Planificação. - Ampliação e redução. - Simetrias			X X X X X X X

ANEXO C – Critérios de avaliação de matemática no ciclo II 2ª etapa

A tabela abaixo contém os critérios determinados pelo MEC, e utilizada como base pelos professores em cada avaliação feita de cada aluno para determinar o desempenho do aluno em cada conteúdo.

CRITÉRIOS DE AVALIAÇÃO 5º ANO
MATEMÁTICA
Identifica a importância da história dos números na composição dos diferentes sistemas de numeração, compreendendo a construção das diferentes bases numéricas e suas propriedades internas? □ Compreende a representação do Sistema de Numeração Decimal, estabelecendo as relações entre unidade, dezena e centena até a classe dos milhões, reconhecendo o valor posicional dos números e realizando agrupamentos e trocas na base dez? □ Realiza agrupamentos para facilitar a contagem, estabelecendo relação entre o símbolo numérico e a quantidade? Compreende antecessor e sucessor na passagem dos milhares para os milhões? □ Realiza composição e decomposição de números até a classe dos milhões? □ Compreende no mínimo os números de 0 a 999999999. (novecentos e noventa e nove milhões, novecentos e noventa e nove mil, novecentos e noventa e nove)? □ Utiliza a regularidade numérica presente em determinadas situações na resolução de problemas? □ Utiliza raciocínio de proporcionalidade na resolução de problemas? □ Resolve problemas de adição utilizando diferentes estratégias como: desenho, estimativa, cálculo mental, algoritmos convencionais ou não? □ Utiliza o algoritmo no cálculo da adição simples e com trocas até a ordem dos milhões? □ Resolve problemas de multiplicação utilizando diferentes estratégias como: desenho, estimativa, cálculo mental, algoritmos convencionais ou não?

 Utiliza o algoritmo no cálculo da multiplicação com mais de dois algarismos no multiplicador?

 Utiliza a idéia de dobro, triplo, quádruplo e outros na resolução de problemas?

 Resolve problemas de subtração utilizando diferentes estratégias como: desenho, estimativa, cálculo mental, algoritmos convencionais ou não?

 Utiliza o algoritmo no cálculo da subtração simples e com reserva até a ordem dos milhões?

 Resolve problemas de divisão utilizando diferentes estratégias como: desenho, estimativa, cálculo mental, algoritmos convencionais ou não?

 Utiliza o algoritmo no cálculo da divisão com mais de dois algarismos no divisor?

 Utiliza a idéia de metade, terça parte, quarta parte e outros na resolução de problemas?

 Utiliza as operações e as relações entre elas para resolver problemas por meio de diferentes estratégias de resolução como desenho, estimativa, cálculo mental, algoritmos convencionais ou não?

Identifica possíveis maneiras de combinar elementos de uma coleção e de contabilizá-los, usando diferentes estratégias de resolução.?

 Lê dados e informações quantitativas e qualitativas apresentados em linguagem gráfica?

 Representa informações quantitativas e qualitativas utilizando a linguagem gráfica?

 Analisa dados e informações quantitativas e qualitativas apresentados em linguagem gráfica?

 Identifica, faz inferências e prevê resultados possíveis em uma situação aleatória?

 Utiliza com compreensão as unidades padrão de medidas de comprimento relacionando os múltiplos e submúltiplos das unidades de medidas mais utilizadas (quilômetro, metro, centímetro e milímetro)?

 Faz conversão e estabelece relações entre as unidades de medida[57]?

 Utiliza o conceito de perímetro e área na resolução de problemas?

 Utiliza com compreensão as unidades padrão de medidas de massa relacionando os múltiplos e submúltiplos das unidades de medidas mais utilizadas (quilograma, grama e tonelada)?

Utiliza com compreensão as medidas de capacidade relacionando os múltiplos e submúltiplos das unidades de medidas mais utilizadas (litro, mililitro)?

 Utiliza com compreensão as medidas de tempo relacionando os múltiplos e submúltiplos?

 Resolve problemas que envolvam medidas de tempo (adição e subtração de horas, minutos e segundos)?

 Utiliza com compreensão as medidas de valor relacionando os múltiplos e submúltiplos?

 Realiza cálculos proporcionais na relação entre as grandezas[58]?

 Reconhece as relações entre as diferentes representações de um número racional e faz uso dessas representações?

 Compreende o conceito de frações de unidade e de quantidade estabelecendo relações entre o todo e suas partes?

 Utiliza a equivalência de frações, com compreensão, na resolução de problemas?

 Utiliza as idéias de operações com frações de uso freqüente[59] na resolução de problemas?

 Realiza operações com números decimais utilizando unidade de medidas[60]
de medidas[60]?

 Utiliza o conceito de porcentagem na resolução de problemas?

 Representa o espaço e a localização de objetos utilizando as
noções topológicas?

 Interpreta a localização em malhas quadriculadas, mapas e em
outras formas de representação, utilizando a linguagem matemática para orientar-se no espaço?

 Comunica a localização em malhas quadriculadas, mapas e
em outras formas de representação, utilizando a linguagem matemática para orientar-se no espaço?

 Representa, em malhas, mapas e maquetes, espaços e objetos com proporcionalidade?

 Identifica em sólidos e modelos de sólidos, corpos redondos e poliedros, classificando-os?

 Realiza planificações, identificando as relações entre as formas tridimensionais e bidimensionais[61]?

 Identifica ângulos retos, agudos e obtusos?

 Identifica os ângulos internos e externos em figuras bidimensionais?

 Representa as figuras bidimensionais de acordo com as suas características?

Amplia e reduz figuras, identificando a proporção entre o objeto real e a sua representação?

 Representa as relações simétricas em objetos e figuras geométricas?

 Utiliza as noções projetivas em representações de objetos e figuras?

 Utiliza as noções euclidianas[62] em representações?