INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAIBA
COORDENAÇÃO DO CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

GLEYDSON DA SILVA TAVARES

MAGISTRO, SISTEMA DE VISUALIZAÇÃO DE BOLETIM ESCOLAR ONLINE

JOÃO PESSOA
2015

Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

Coordenação do curso superior de tecnologia em sistemas para internet

MAGISTRO, SISTEMA DE VISUALIZAÇÃO DE BOLETIM ESCOLAR ONLINE

Relatório de Estágio Supervisionado apresentado à disciplina Estágio Supervisionado do Curso Superior de Tecnologia em Sistemas para Internet do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba como requisito parcial para obtenção do grau de Tecnólogo em Sistemas para Internet.

Orientador: Luiz Carlos Rodrigues Chaves
Supervisor: Telma Maria Pereira de Medeiros Rodrigues
Coordenador(a) do Curso: Valéria Maria Bezerra Cavalcanti
Empresa: Lyceu Paraibano
Período: 03/03/2015 a 15/06/2015

João Pessoa
2015

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAIBA
COORDENAÇÃO DO CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA INTERNET

GLEYDSON DA SILVA TAVARES

Supervisor: Telma Maria Pereira de Medeiros Rodrigues
Lyceu Paraibano

Coordenadora do CST em Sistemas para Internet:
Profª Valéria Maria Bezerra Cavalcanti

Professor Orientador:
Prof. Luiz Carlos Rodrigues Chaves

Estagiário:
Gleydson da Silva Tavares

João Pessoa, 23 de Novembro de 2015

Agradecimentos

A Deus por ter me dado saúde e força até esse momento. A este instituto, seu corpo docente, direção e administração que oportunizaram a janela que hoje vislumbro um horizonte superior, eivado pela acendrada confiança no mérito e ética aqui presentes. Ao meu orientador Luiz Carlos Rodrigues Chaves, pelo suporte no pouco tempo que lhe coube, pelas suas correções e incentivos. Aos meus pais e irmão, pelo amor, incentivo e apoio incondicional. E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

"Escolha um trabalho que ama
e não terás que trabalhar um único dia em sua vida.
(Confúcio)

Resumo

Este relatório tem como objetivo descrever as atividades realizadas durante o Estágio Curricular Supervisionado realizado na escola estadual de ensino médio Lyceu Paraibano, localizada em João Pessoa, PB. As atividades foram exercidas no período de 03 de Março de 2015 a 15 de Junho de 2015. O estágio teve como objetivo aplicar, no desenvolvimento de um sistema de visualização de Boletim escolar online, os conceitos teóricos e práticos aprendidos durante o curso de Sistemas para Internet do Instituto Federal de Ciência e Tecnologia da Paraíba (IFPB) e verificar até que ponto esses conceitos são realizáveis na resolução de problemas do cotidiano escolar.

Palavras-chaves: Desenvolvimento de Sistemas, Boletim, Secretaria de Estado da Educação, Educação, Escola.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

MEC	Ministério da Educação e Cultura
UML	Unified Modeling Language (Modelagem de Linguagem Unificada)
MVC	Model, View, Controller
NAP	Núcleo de apoio pedagógico
AJAX	Asynchronous Javascript and XML

Lista de Códigos

Sumário

1 Introdução

Neste documento serão demonstradas as atividades de estágio realizadas na escola estadual de ensino médio Lyceu Paraibano, expondo o conteúdo técnico e prático que serviram para a criação de um sistema de visualização de Boletim escolar online denominado Magistro. O estágio foi desenvolvido no período de 03/03/2015 a 15/06/2015 sob orientação acadêmica de Luiz Carlos Rodrigues Chaves e supervisão técnica da diretora escolar Telma Maria Pereira de Medeiros Rodrigues. O foco do trabalho inicialmente foi fazer um levantamento dos possíveis requisitos funcionais do sistema junto aos funcionários da secretaria da escola e após o levantamento desses requisitos desenvolver o sistema. O restante do relatório aborda com mais detalhes a descrição das atividades, descrevendo as atividades realizadas, as ferramentas utilizadas e a metodologia usada para criação do sistema.

1.1 Objetivo

O estágio teve como objetivo desenvolver um sistema de visualização de boletim escolar online, onde os professores, pais e alunos pudessem ter um acompanhamento das notas, como também a geração automática de alguns documentos da escola que dependem dessas notas.

1.2 A Empresa

A empresa concedente do estágio é a escola estadual de ensino médio Lyceu Paraibano localizada na Av. Pres. Getúlio Vargas - Centro, João Pessoa - PB. O Lyceu Paraibano é reconhecido como a escola pública com mais destaque do estado da Paraíba, pela qualidade dos seus professores ou por ser a escola de grandes nomes da história do nosso Estado.

O Lyceu Paraibano foi fundado pela Lei n o 11, de 24 de Março de 1836, e permaneceu durante 117 anos como o único estabelecimento público de ensino secundário na Paraíba. Funcionou por muito tempo no antigo edifício da tesouraria da fazenda, correios e telégrafos, da assembleia legislativa e que hoje abriga a academia de direito da UFPB. Em 1936 cem anos depois da sua fundação, definitivamente o Lyceu ganhou sede própria na avenida Getúlio Vargas no cento da capital Paraibana na administração do então governador Argemiro de Figueiredo.

De acordo com Censo de 2014 a escola conta com mais de 2111 alunos matriculados

nos 3 turnos de ensino, 163 funcionários, 18 computadores para os alunos e 8 computadores para uso administrativo.

Devido a grande quantidade de alunos e de documentos gerados no final de cada ano letivo a direção escolar junto com o núcleo de apoio pedagógico (NAP) achou necessário o desenvolvimento de um sistema que possibilitasse a visualização das notas dos alunos, a frequência escolar e a geração automática de alguns documentos que dependem dessas notas.

1.3 Descrição Geral das Atividades

Nessa seção são apresentadas as atividades realizadas no decorrer do estágio que estão em ordem cronológica para melhor organização.

- Levantamento dos requisitos funcionais e não funcionais;
- Modelagem do sistema usando UML;
- Desenvolvimento do sistema;
- Testes de funcionalidade.

1.4 Organização do relatório

Além desse capítulo, o relatório está dividido em outros 3 capítulos, brevemente descritos a seguir:

- **Capitulo 2** - Embasamento teórico: Nesse capítulo são descritas as tecnologias adotadas no desenvolvimento do sistema, a metodologia e as ferramentas.
- **Capitulo 3** - Atividades realizadas: Nesse capítulo são descritas as atividades realizadas no estágio.
- **Capitulo 4** - Conclusão: Capítulo com o relato das experiências adquiridas, objetivos alcançados, trabalhos futuros e considerações finais.

2 Embasamento Teórico

Neste capítulo são descrito os conceitos e tecnologias envolvidas em todo processo de desenvolvimento do sistema Magistro.

2.1 Scrum

Para o desenvolvimento do Magistro foi adotado o framework ágil de desenvolvimento de software Scrum. O Scrum foi escolhido por ser mais simples em relação as outras metodologias ágeis e de fácil adaptação já que todo o sistema foi desenvolvido por apenas um desenvolvedor.

O Scrum foi concebido em 1993 por Jeff Sutherland da necessidade de encontrar uma metodologia que abordasse o problema do desenvolvimento de software de uma forma não tradicional (FERREIRA et al., 2005), em 1995 Ken Schwaber refinou a metodologia baseado na sua própria experiência no desenvolvimento de sistemas e processos.

O Scrum é um conjunto de atividades e práticas de gestão que devem ser seguidas para garantir que no final do projeto seja entregue um produto como especificado pelo cliente. Ela segue o paradigma iterativo e incremental onde a cada iteração pequenas partes do sistema são entregues. Isso é importante pois garante que teremos partes funcionais do sistema e essas passarão por todas as atividades até a implantação. As principais características do SCRUM segundo Ferreira et al. (2005) são:

- é um processo ágil para gerenciar e controlar o desenvolvimento de projetos;
- é um wrapper para outras práticas de engenharia de software; é um processo que controla o caos resultante de necessidades e interesses conflitantes;
- é uma forma de aumentar a comunicação e maximizar a cooperação;
- é uma forma de detectar e remover qualquer impedimento que atrapalhe o desenvolvimento de um produto;
- é escalável desde projetos pequenos até grandes projetos em toda empresa.

2.1.1 Vocabulário utilizado pelo SCRUM:

Time Scrum: No Time Scrum não existe um líder para ditar as regras do desenvolvimento, o time é autogerido. Por isso, a opinião de todos os envolvidos é importante para o progresso do projeto.

Scrum Master: O Scrum Master não é o líder do time já que o scrum não possui um, mas sim a pessoa com bom conhecimento técnico em Scrum que organiza o projeto facilitando o andamento e garantindo que todos cumpram suas tarefas.

Product Owner: É o responsável por traduzir as histórias do cliente em requisitos para que a equipe possa trabalhar no Sprint.

Equipe de desenvolvimento: São aqueles que participarão na codificação do produto.

Eventos: São pequenas conversas com tempo fixo, objetiva e ágil.

Sprint: É um ciclo de desenvolvimento ou em outras palavras um tempo determinado para entregar uma tarefa. No final de cada Sprint é entregue algo concreto.

Reunião de planejamento: Reuniões realizadas para definir os objetivos do Sprint. Para calcular a quantidade máxima de horas de cada reunião multiplicasse cada semana do Sprint por 2, se um Sprint tem duração de 1 semana, por exemplo, a reunião terá no máximo 2 horas de duração.

Reunião diária: Reunião com no máximo 15 minutos para revisar o andamento do projeto.

Revisão da Sprint: Revisar os itens desenvolvidos e mostrar as novas funcionalidades.

Retrospectiva da Sprint: Serve para fazer um balanço geral da Sprint e verificar o que funcionou bem, as melhorias que tem que ser feitas, os erros e o que será feito para melhorar.

Product backlog: É uma lista com tudo que o cliente desejaria como funcionalidade. É criado pelo product owner e sofre mudanças a cada solicitação do cliente.

Sprint backlog: É uma lista com o que deve ser feito na próxima Sprint.

2.1.2 Scrum Solo

Scrum solo não é uma nova metodologia, é apenas um denominação dada a uma adaptação do Scrum para projetos que possuem apenas um desenvolvedor. Assim, a pessoa envolvida no projeto tem a liberdade de remover algumas coisas desnecessárias do Scrum quando se está trabalhando sozinha.

Durante o desenvolvimento do sistema Magistro, foi utilizada a seguinte adaptação do Scrum:

- Uma lista das funcionalidades do sistema (Product backlog);
- Algumas funcionalidades selecionadas para desenvolver (Sprint backlog);
- Controle das atividades utilizando um quadro Kanban;

- Cada produto entregue no final da Sprint era validado por funcionários da secretaria da escola;
- Reuniões com o pessoal da secretaria para reavaliar o Product backlog.

2.2 JavaScript

JavaScript é uma linguagem de programação criada em 1995 por Brendan Eich na Netscape (GOODMAN, 2001). Foi utilizada inicialmente no navegador Netscape Navigator 2.0 permitindo a criação de conteúdos dinâmicos a partir da manipulação de componentes. Com ela, podemos manipular os componentes diretamente no browser sem a necessidade de fazer requisições no servidor.

Apesar de ser conhecida por dar dinamicidade a conteúdos web, a linguagem de programação JavaScript não se limita apenas ao mundo web, ela pode ser também utilizada em diferentes tipos de ambientes.

As principais características da linguagem segundo Munzlinger (2011) são:

- **Interpretada:** JavaScript não necessita ser compilada, pois é uma linguagem interpretada diretamente pelo navegador.
- **Case sensitive:** Em JavaScript deve-se respeitar letras maiúsculas e minúsculas.
- **Fracamente tipada:** JavaScript é chamada de fracamente tipada pois ao declarar as variáveis não definimos qual o tipo da variável(inteira, ponto flutuante, array, booleana, etc). Assim, podemos armazenar em uma variável a informação que desejamos, pois não tem tipo "definido"na declaração.
- **Dinamicamente tipada:** Toda linguagem fracamente tipada é também dinamicamente tipada. Isso significa que a variável vai se tornar de um determinado tipo no momento da atribuição do conteúdo. Se o tipo do conteúdo mudar durante a execução do programa a variável vai mudar de tipo dinamicamente.
- **Estruturada ou orientada a objetos:** Originalmente a linguagem foi criada seguindo o paradigma de programação estruturada, mas também pode ser programada no paradigma de programação orientada a objetos.
- **Acionamentos:** Scripts podem ser acionados por eventos(passar o mouse, clicar, tempo, etc) ou automaticamente na sequência em que o navegador interpreta o conteúdo da página e localiza a linha de comando.

2.3 jQuery

JQuery é um framework JavaScript que nada mais é que uma coleção de funções pré-definidas e testadas. Foi lançado pela primeira vez em 2006 e hoje é usado por mais de 67.9% dos sites mais visitados do mundo.[1]

De acordo com Silva (2008) a biblioteca fornece dinamismo e interatividade às páginas web com o intuito de tornar agradável a experiência do usuário com a aplicação,

[1] <http://w3techs.com/technologies/overview/javascript_library/all>

além de fornecer diversas funcionalidades ao desenvolvedor para facilitar a criação de scripts. Esses scripts visam incrementar, de forma progressiva e não obstrutiva, a usabilidade, a acessibilidade e o design, enriquecendo a experiência do usuário.

2.4 Grails

O framework web Grails, segundo Anselmo (2010), é um framework completo para a máquina virtual Java, por tirar proveito da linguagem de programação Groovy e fornecer uma experiência de desenvolvimento produtivo e de fluxo alinhado.

O Grails foi criado em 2006 influenciado diretamente pelo surgimento do Ruby on Rails. Grails utiliza o paradigma de programação por convenção o que reduz o tempo gasto com configurações deixando assim o programador focado no desenvolvimento do sistema. Possui mais de 1000 plugins que podem ser baixados no endereço https://grails.org/plugins/.

Algumas vantagens oferecidas pelos Grails segundo FRANCO (2011) são:

- uma camada para mapeamento objeto-relacional fácil de usar construída no Hibernate;
- Uma tecnologia de visão expressiva, chamada Groovy Server Pages (GSP);
- Uma camada de controle construída no Spring MVC;
- Um ambiente em linha de comando construído no Groovy: Gant.
- Um container TomCat integrado que é configurado para recarregar quando necessário;
- Injeção de dependência com o container Spring embutido;
- Suporte à internacionalização (i18n) construída sobre o conceito MessageSource do núcleo do Spring;
- Uma camada de serviços transacionais construída na abstração de transações do Spring.

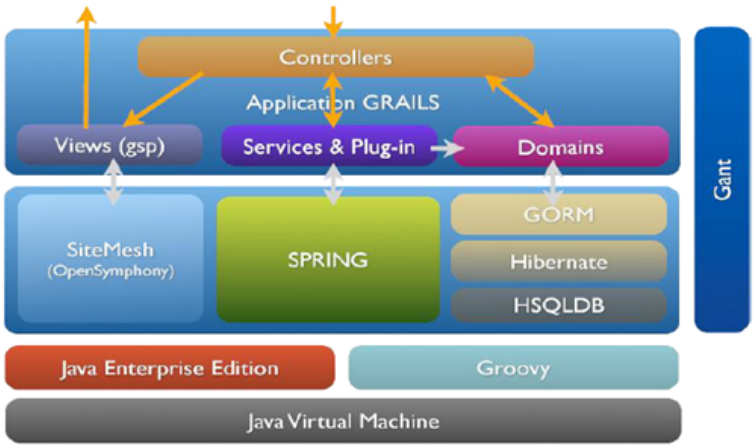
Seguindo o paradigma de convenção sobre configuração o Grails possui uma estrutura de diretórios fixa onde cada arquivo tem seu lugar pré-definido da seguinte forma:

- **Grails App** Diretório principal de um projeto Grails
  - Conf – onde fica localizado os arquivos de configuração.
  - Controllers – os arquivos controladores seguindo o padrão MVC.
  - Domain – os arquivos de domínio seguindo o padrão MVC.
  - i18n – internacionalização.
  - services – arquivos de serviços, remove a lógica de negócio dos.
  - Views – arquivos .GSP que tratam da camada de visualização.

A versão mais recente do Grails é a 3.0.9 e utiliza a versão 2.4 da linguagem Groovy.

Abaixo, a figura 2.4 mostra a arquitetura do framework Grails.

Figura 1: Arquitetura do Grails

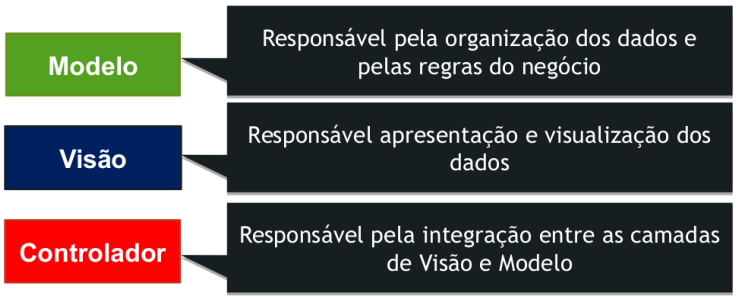


Fonte: http://eusoudev.blogspot.com.br/2014/03/visao-geral-do-grails-framework.html

2.4.1 Padrão MVC

No padrão MVC(Model-View-Controller) a aplicação é dividida em 3 camadas, figura 2.4.1.

Figura 2: Camadas do padrão MVC



Fonte: http://www.devall.com.br/blog/show/1358

A camada de controle é a responsável por processar as ações do usuário e enviar essas para as camadas de domínio ou de visualização. No Grails essa camada é uma intermediação entre as camadas de modelo e visualização, sendo uma prática ruim colocar a lógica do negócio nessa camada, para isso, Grails provê a pasta Services que é onde deve ficar os serviços com as lógicas de negócio da aplicação. A camada de modelo é a camada que contém as classes de modelo que são classes que possuem a função de definir os atributos e as regras de negócio. A camada de visualização é a camada responsável por fazer a ligação entre o usuário e a aplicação.

Em Grails seguindo a convenção as requisições a uma determinada ação é feita obedecendo a seguinte convenção de URL: http://localhost:8080/controlador/acao onde *controlador* é um controlador qualquer da nossa aplicação e ação é uma função ou closures presente nesse controlador, a página .gsp por convenção deve possuir o mesmo nome da

ação, no nosso caso acao.gsp.

2.5 Groovy

Groovy é uma linguagem ágil e dinâmica para a plataforma Java, com muitas ferramentas inspiradas em linguagens como Python, Ruby e Smalltalk, tornando-as disponíveis aos programadores, usando uma sintaxe próxima do Java (KOENIG, 2011).

Para Koenig (2011), a linguagem possui as facilidades de outras linguagens como Ruby e Python, e o poder e estabilidade do Java. O código Groovy é compilado em bytecodes Java, isso quer dizer que você pode escrever em Groovy e em Java diminuindo assim a curva de aprendizagem.

Em Groovy, os seguintes pacotes Java são importados por *default*:

- java.lang.*
- java.util.*
- java.io.*
- java.net.*
- groovy.lang.*
- groovy.util.*
- java.math.BigInteger
- java.math.BigDecima

Não existe tipos primitivos em Groovy, tudo é um objeto, métodos gets e sets são criados dinamicamente e classes e métodos são declarados como público.

Alguns recursos presentes em Groovy:

- Clousures
- Tipagem dinâmica e estática
- Expressões dento de Strings
- Interpolar Strings
- Alteração de código em tempo de execução
- Construtores específicos para coleções

2.6 UML

Unified Modeling Language (UML) é uma linguagem visual utilizada para modelar sistemas computacionais por meio do paradigma de Orientação a Objetos. Essa linguagem se tornou, nos últimos anos, a linguagem-padrão de modelagem de software adotada internacionalmente pela indústria de Engenharia de Software (GUEDES, 2014). A UML disponibiliza uma forma padrão de modelagem de projetos de sistemas, incluindo seus aspectos conceituais tais como processos de negócios e funções de sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, processos de banco de dados e componentes de software reutilizáveis (SILVA; VIDEIRA, 2001)

A linguagem UML é composta por muitos elementos de modelo que representam as diferentes partes de um sistema de software. Os elementos UML são usados para criar diagramas, que representam uma determinada parte, ou um ponto de vista do sistema. No UML pode-se construir os seguintes diagramas de acordo com Hensgen (2015):

- **Diagrama de Caso de Uso** mostra os atores do sistema, casos de uso e seus relacionamentos;
- **Diagrama de Classe** mostra as classes os relacionamentos entre elas e os atributos;
- **Diagrama de Sequência** mostra os objetos e a sequência das chamadas de método feitas a outros objetos.
- **Diagrama de Colaboração** mostra os objetos, seus relacionamentos, dando ênfase aos objetos que participam da troca de mensagens;
- **Diagrama de Estado** mostra os estados, mudanças dos estados e eventos em determinado objeto ou uma parte do sistema;
- **Diagrama de Atividade** mostra as atividades e as mudanças de uma atividade para outra de acordo com os eventos ocorridos em alguma parte do sistema;
- **Diagrama de Componente** mostra os componentes de programação em alto nível;
- **Diagrama de Distribuição** mostra as instâncias dos componentes e seus relacionamentos.

2.7 Draw.io

Draw.io [2] é uma ferramenta gratuita para criação de gráficos, diagramas, UML, etc. Essa ferramenta permite que o usuário salve todo o seu trabalho no Google Drive,

[2] <https://www.draw.io>

Dropbox, no próprio computador ou no navegador web com o uso de plugins, permitindo assim termos uma cópia do nosso trabalho localmente ou na nuvem.

A ferramenta possui uma interface amigável e simples, permite a exportação do projeto em vários formatos como: JPG, PNG, PDF, SVG, HTML e XML.

A interface do draw.io pode ser vista na figura 3 a seguir:

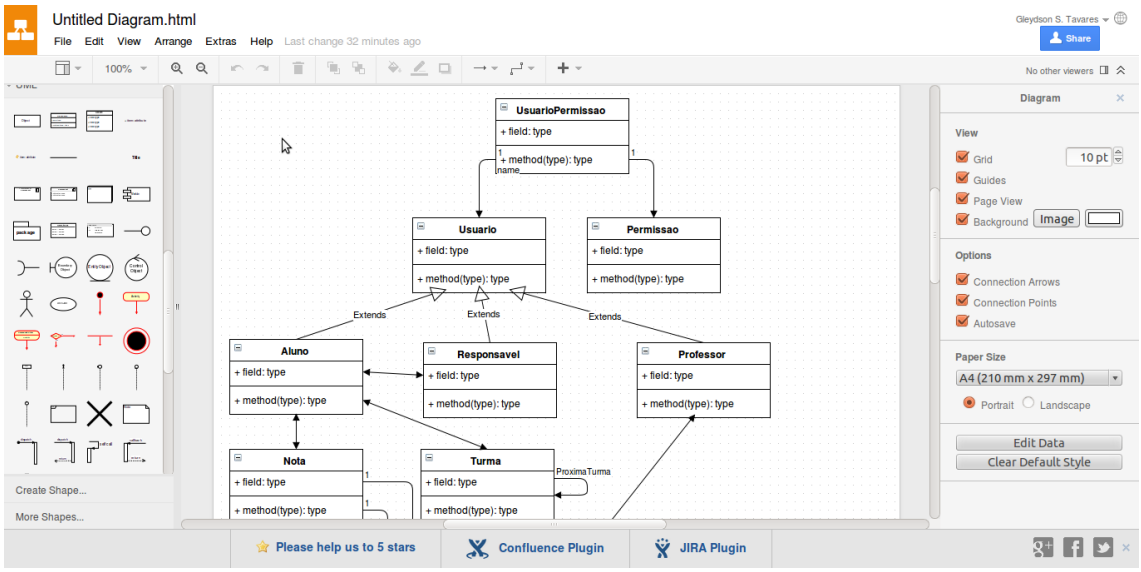


Figura 3: Interface do Draw.io

3 Atividades desenvolvidas

Nesse capítulo são dispostas as atividades realizadas durante o período do estágio na escola estadual de ensino médio Lyceu Paraibano.

3.1 Levantamento de requisitos funcionais

A metodologia ágil escolhida para o desenvolvimento do sistema foi a Scrum. Geralmente essa metodologia é adotada por uma equipe de desenvolvimento, mas no caso como existiria apenas um desenvolvedor foi decidido suprimir algumas características padrão do *Scrum*.

Sendo funcionário da escola, atuando na secretaria da mesma e assim estando por dentro de todas as atividades, fui durante o desenvolvimento do projeto o cliente (product owner). Sendo assim, também inseria algumas funcionalidades que o sistema poderia ter no *product backlog* e validava-as quando com outros possíveis usuários do sistema (professores, alunos e diretores).

A primeira atividade depois de uma reunião com a direção e alguns professores foi criar a lista de possíveis funcionalidades que haveria no sistema *product bakclog*. Cada nova funcionalidade que aparecia era colocada em uma tabela conforme a Tabela1 com os campos ID, Título Descrição e Sugerido por, conforme fossem aparecendo novas funcionalidades a tabela era atualizada. Logo após, foi feita uma seleção das funcionalidades mais importantes que entrariam no *sprint backlog*. Para cada *log* de implementação foi estabelecido a duração máxima de uma semana e as reuniões entre os membros da equipe de desenvolvimento que ocorrem diariamente segundo o padrão Scrum não existiam. Um quadro Kanban foi o único artefato utilizado nessa etapa. Nesse quadro existia as funcionalidades a serem desenvolvidas e os *backlogs* separados por categorias: *To do, In Progress e Done*.

O produto entregue(as funcionalidades implementadas no *sprint backlog*) eram então validadas por possíveis usuários do sistema.

Algumas regras de negócio foram elicitadas após a primeira reunião com a direção, dentre as principais regras estão:

- alunos, professores e pais só podem acessar conteúdos do respectivo perfil;
- professores podem dar aula em várias turmas e em disciplinas diferentes;
- alunos e professores podem mudar de turma;

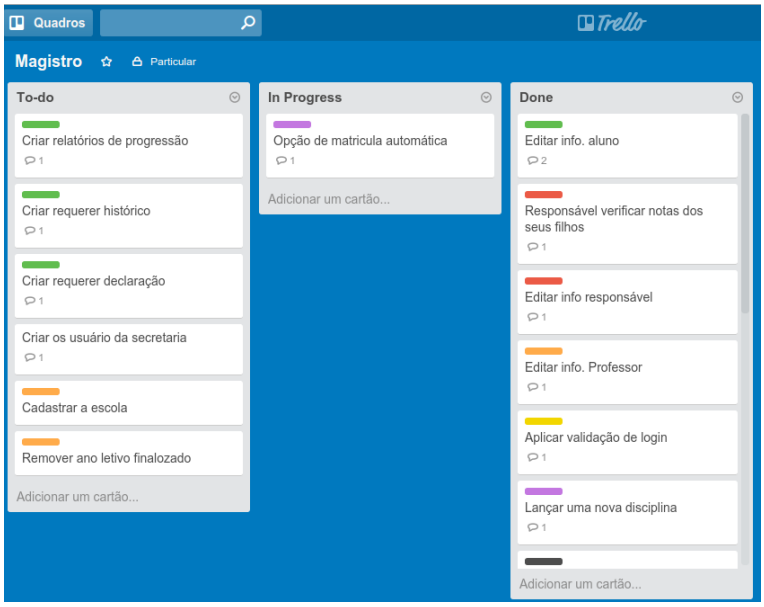


Figura 4: Quadro Kanban usando o trello

- o administrador é o responsável por cadastrar alunos, responsáveis, professores e administrar o sistema;
- o aluno caso maior de 18 anos pode ser responsável por si mesmo;
- as notas só serão lançadas após o início do ano letivo e antes do término do ano letivo;
- um ano letivo só é encerrado quando todas as notas forem lançadas pelos professores;
- um novo ano letivo só é iniciado após o encerramento do ano letivo anterior.

Tabela 1: Tabela de funcionalidades

3.2 Modelagem do sistema usando UML

Após o levantamentos dos requisitos do sistema, foi feita uma análise desses requisitos para a criação do diagrama de classes do sistema usando Unified Modeling Language (UML).

Segundo Furlan (1998), diagrama de classe trata-se de uma estrutura lógica estática em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos de modelo, como classes, tipos e seus respectivos conteúdos e relações. A figura 3.2 mostra o diagrama de classe do Magistro.

Devido ao curto tempo para criação do sistema e para evitar possíveis surpresas durante o desenvolvimento, foi dada uma grande ênfase nessa atividade para entender de que forma o projeto seria conduzido.

O diagrama de classes é utilizado para se ter uma visão geral das classes e dos seus relacionamentos. A figura 3.2 exibe o diagrama de classe do Magistro contendo as 11 classes e relacionamento entre elas.

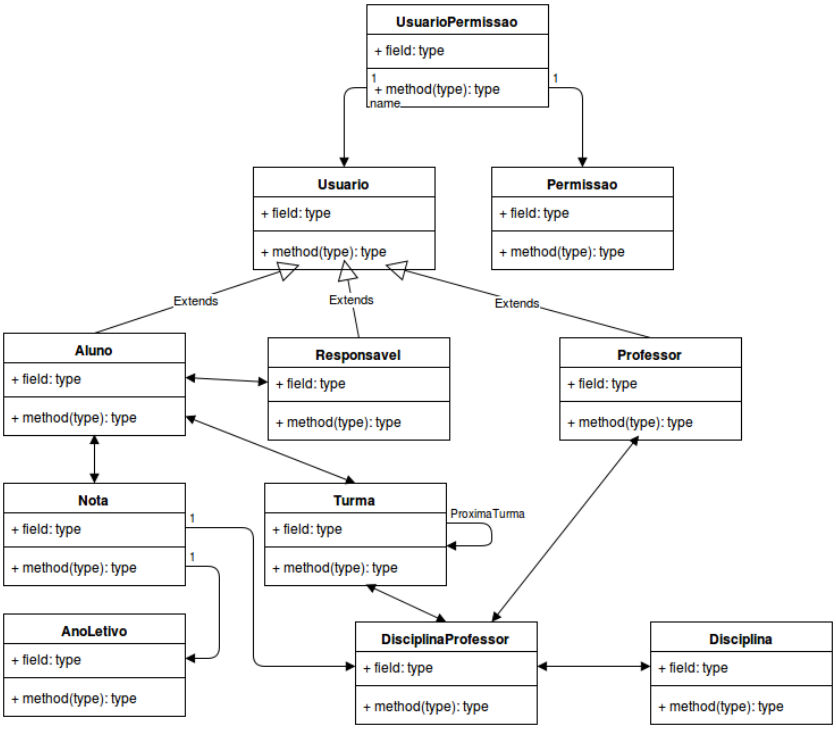


Figura 5: Diagrama de classe do Magistro

A seguir é apresentada uma breve descrição das classes do diagrama (Figura 3.2):

- **Usuário:** Classe que contêm os dados referentes a todos os usuários do sistema;
- **Permissão:** Classe gerada pelo plugin springSecurityCore;
- **UsuarioPermissao:** Classe que contêm os dados referentes a um usuário e sua permissão;
- **Responsável:** Classe que contêm os dados referentes ao responsável pelo aluno;
- **Aluno:** Classe que contêm os dados referentes ao aluno;
- **Turma:** Classe que contêm os dados referentes as turmas;
- **Nota:** Classe que contêm os dados referentes as notas do aluno;
- **AnoLetivo:** Classe que contêm os dados referentes ao ano letivo;

- **Disciplina:** Classe que contêm os dados referentes as disciplinas;
- **Professor:** Classe que contêm os dados referentes aos professores;
- **DisciplinaProfessor:** Classe que contêm os dados referentes a disciplina e ao professor que ministra a disciplina;
- **Usuário:** Classe que contêm os dados referentes a todos os usuários do sistema;

3.3 Desenvolvimento do sistema

Nessa seção são descritas as principais funcionalidades desenvolvidas do sistema.

3.3.1 Autenticação dos usuários

O Magistro possui autenticação para todos usuários que acessam o sistema. Essa autenticação no Grails pode ser feita de forma convencional através de filtros e *sessions* ou através de plugins.

No desenvolvimento do Magistro foi adotado o plugin *spring-security-core* [1]. O spring-security-core é uma biblioteca com várias funções baseada no *Spring Framework*, essas funções fornecem controle de acesso baseado em um usuário e em sua respectiva permissão.

No Magistro, cada usuário possui sua permissão, as permissões desenvolvidas para o sistema foram as seguintes: aluno, professor, responsável e admin. Abaixo o trecho de código 3.1 mostra como é realizado o controle de acesso para o método create do controlador de alunos, através da anotação @Secured do *spring-security-core* é possível passar como parâmetro as permissões que desejamos.

Código 3.1: Autenticação do usuário

```
@Secured(['ROLE_ADMIN'])
def create() {
    def alunoInstance = new Aluno(params)
    respond alunoInstance
}
```

Apenas o administrador do sistema possui a permissão de cadastrar o ano letivo, alunos, professores, responsáveis e gerenciar todo o sistema. Alunos, professores e responsáveis também possuem funcionalidades do respectivo perfil.

[1] <http://grails.org/plugin/spring-security-core>

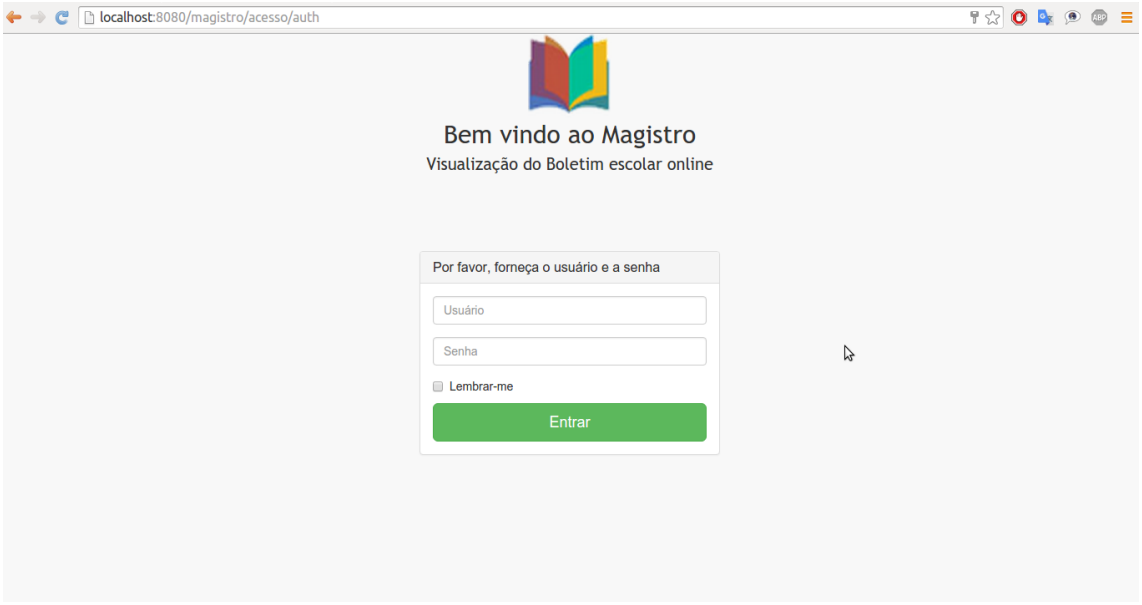


Figura 6: Tela inicial de login

3.3.2 Geração do boletim editável

Ao acessar a funcionalidade de lançar notas, o professor escolhe uma turma onde ele ministra a aula, a disciplina e a etapa. É exibido para ele uma tabela com os campos editáveis das notas e das faltas como nos mostra a figura 3.3.2.

Para essa funcionalidade de edição dinâmica das tabelas foi utilizado um plugin *jQuery* chamado *jqGrid* [2]. Uma vez que os dados são manipulados no lado do cliente o plugin utiliza *Ajax* para fazer a chamadas no lado do servidor e atualizar as notas dos alunos.

Para criar uma tabela editável usando o *jqGrid* é necessário primeiramente criarmos uma tabela *HTML* com uma ID e com a classe específica do *jqGrid* como mostra o trecho de código 3.2

Código 3.2: Tabela html jqGrid

```
<table id="notas_alunos_list" class="scroll␣jqTable"></table>
```

Após isso, criamos uma função *javaScript* para manipular nossa tabela e atualizar as notas no lado do servidor usando *Ajax*. Código 3.3.

Código 3.3: função javaScript jqGrid

```
$(document).ready(function () {
    $("#notas_alunos_list").jqGrid({
  url:'alunosDaTurmaDisciplina',
  cellurl:'editarNota',
```

[2] <http://www.trirand.com/blog/>

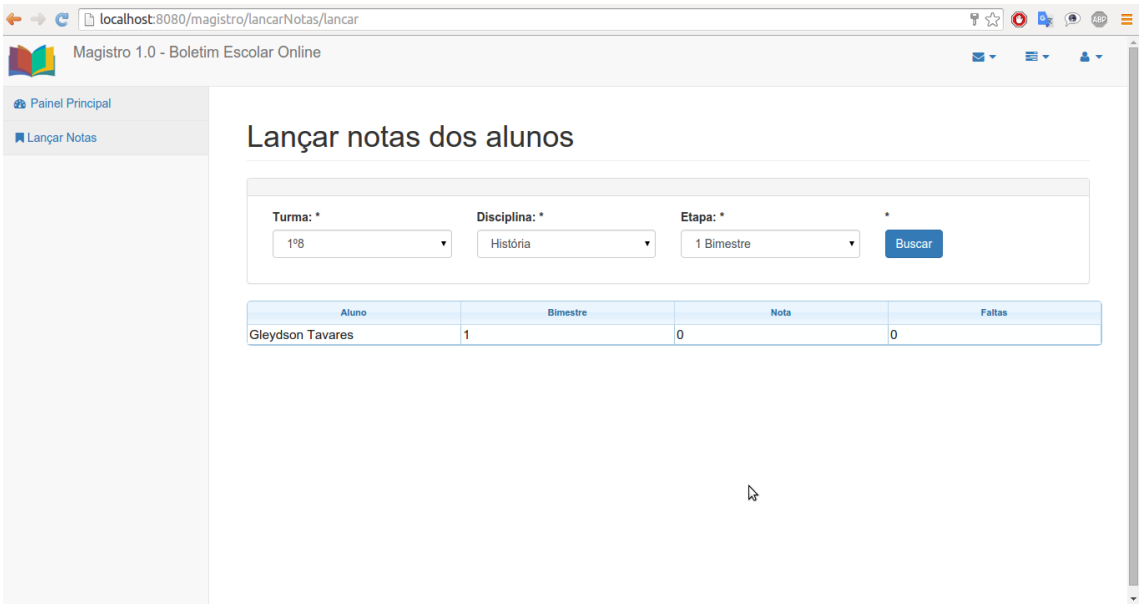


Figura 7: Tabela editável para lançamento das notas

```
    datatype: "json",
    cellEdit:true,
    colNames:['Aluno','Bimestre','Nota','Faltas'],
    colModel:[
      {name:'alunoNome'},
      {name:'bimestre'},
      {name:'nota', editable:true,editrules:{required:true,number:t
      {name:'faltas',editable:true,editrules:{integer:true}}
  ],
});
```

Então temos os seguintes campos da nossa função:

- **url:** url que retorna como resposta um json;
- **cellurl:** url que será chamada para atualizar os dados;
- **datatype:** o tipo de dado retornado ao acessar a url, pode ser um xml, json,jsonp,array, xmlstring, jsonstring e script;
- **celledit**: se a tabela será editável;
- **colNames**: o nome das colunas;
- **colModel**: o dos atributos do nosso model especificando quais deles serão editáveis, as validações e o tipo;

Após as notas estarem preenchidas, é feita uma requisição *Ajax* ao controlador responsável por atualizar as notas dos alunos. O código 3.4 mostra como é feito esse procedimento.

Código 3.4: Método responsável por atualizar as notas

```
@Secured(['ROLE_PROFESSOR'])
def editarNota(){
  def nota = null
  switch(params.oper) {
    case 'edit':
      nota = Nota.findById(params.id)
      if(params.nota){
        nota.nota = params.double('nota')
      }
      if (params.faltas) {
        nota.faltas = params.int('faltas')
      }
      if (!nota.save(flush:true)) {
        nota = null
      }
    break
  }
  render nota as JSON
}
```

Segundo o código, é feito um *switch* em *params.oper* para saber qual tipo de operação está sendo realizada, caso seja uma operação de edição então é buscado a nota de forma dinâmica pelo seu ID e a nota é atualizada pelo seu novo valor passado. No fim, ela é salva e é retornado um *JSON* como resposta ao cliente.

3.3.3 Integração com o Twitter Bootstrap

A interface do Magistor é totalmente responsiva garantindo que o mesmo possa ser utilizado em dispositivos com diferentes resoluções de tela. O Twitter Bootstrap [3] foi a biblioteca escolhida para fazer a responsividade do sistema.

Para integrar o Twitter Bootstrap com o Magistro foi mais conveniente utilizar um plugin Grails chamado twitter-bootstrap [4]. Para adicionar esse plugin no nosso projeto foi preciso apenas editar o arquivo *BuildConfig;groovy* que é encontrado na pasta *grails-*

[3] <http://getbootstrap.com/>
[4] <https://grails.org/plugin/twitter-bootstrap>

app/conf/, inserir a seguinte linha em plugins: *runtime ':twitter-bootstrap:3.3.5'* e adicionar os *requeriments* no *Asset-Pipeline* do projeto.

A imagem 3.3.3 mostra a função de cadastrar um ano letivo de forma responsiva em um dispositivo com resolução 480 x 800.

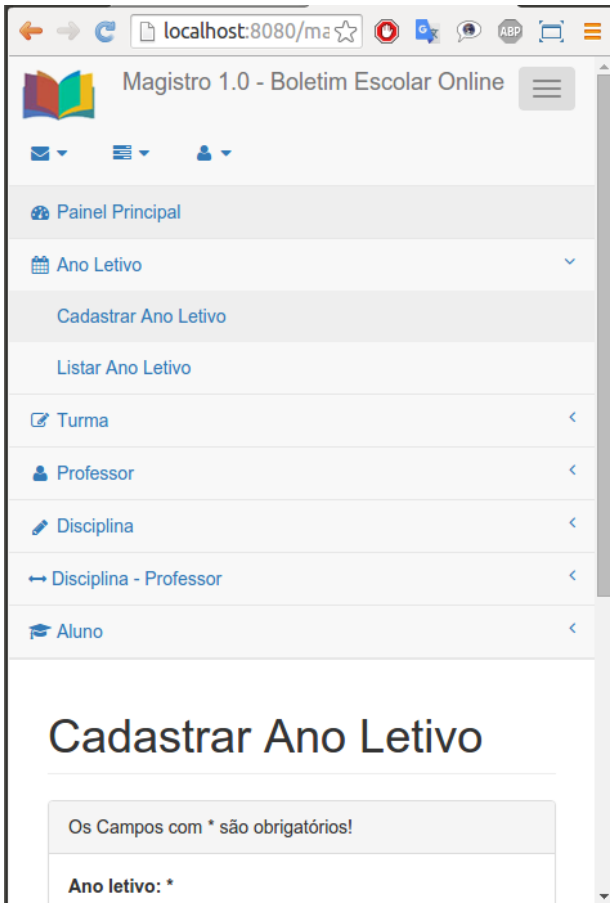


Figura 8: Cadastro do ano letivo responsivo.

3.3.4 Buscas Ajax com formulários remotos.

Por padrão, Grails já oferece para nós suporte nativo a Ajax através de um pequeno conjunto de tags, tornando a tarefa de trabalhar com *Ajax* bem trivial Weissmann (2015).

No Magistro, o administrador possui a opção de buscar alunos,professores, pais, turmas, etc. Para fazer essa busca foi utilizado uma tag do Grails chamada *g:formRemote* que dispara uma requisição Ajax.

O exemplo de código 3.8 mostra o uso da tag *formRemote* para buscar um aluno pelo nome.

Código 3.5: formRemote para buscar aluno pelo nome

```
<g:formRemote name="buscarAluno" update="resultado"
      url="[controller:'aluno',action:'buscarAluno']">
```

```
        Nome: <g:textField name="nome"/>
        <input type="submit" value="Pesquisar"/>
    </g:formRemote>

    <div id="resultado"></div> #template
```

No atributo update é definido um identificador da tag HTML que vai receber o retorno da consulta. O atributo url define qual a action que deverá ser executada Weissmann (2015).

Abaixo o método buscarAlunos do controlador Aluno.

Código 3.6: action buscarAlunos

```
def buscarAluno(){
      def alunos = Aluno.findAllByNomeLike("%${params.nome}%")
      render(template:'resultado',model:[alunos:alunos])
}
```

E o template que exibe o resultado da consulta.

Código 3.7: template resultado

```
<table>
 <thead>
   <th>Nome</th>
 </thead>
 <tbody>
      <g:each in="${alunos}" var="aluno">
       <tr>
         <td>Aluno: "${aluno.nome}"</td>
       </tr>
      </g:each>
 </tbody>
</table>
```

3.3.5 Geração de gráficos de desempenho escolar

Quando o aluno acessa a sua página é possível por meio da aba progressão visualizar seu desempenho escolar durante o ano letivo ou se preferir pode também visualizar seu desempenho em relação a anos anteriores.

Para criação desses gráficos foi utilizada uma ferramenta da Google chamada *Google chart tool* [5] que permite a visualização de gráficos em linhas ou até gráficos mais complexos.

Para utilizar essa ferramenta foi preciso adicionar algumas bibliotecas javaScript na página web, fazer uma requisição Ajax para retornar as notas do aluno agrupadas por bimestre em formato Json, selecionar a opção de gráfico a ser utilizada e criar um gráfico com uma * * atribuindo uma *Id* a ser utilizada pelo *Google Chart tool*.

O trecho de código 3.8 exibe o código utilizado para criação do gráfico.

Código 3.8: Código para o gráfico de desempenho escolar

```
<script type="text/javascript">
 google.load('visualization', '1', {'packages':['corechart']});
 google.setOnLoadCallback(drawChart);
 function drawChart() {
   var jsonData = $.ajax({
       url: "${createLink(controller:'aluno',action:'getNotas')}",
       dataType: "json",
       async: false
       }).responseText;
   var data = new google.visualization.DataTable(jsonData);
   var chart = new google.visualization.LineChart(
   document.getElementById('chart_div'));
   chart.draw(data, {width: 900, height: 400});
 }
 </script>
```

A figura 3.3.5 exibe o gráfico de desempenho anual do aluno.

3.3.6 Testes funcionais com GEB

Os testes funcionais foram feitos para avaliar o comportamento da aplicação durante o desenvolvimento. Grails possui alguns plugins que auxiliam nessa parte de testes funcionais.

O plugin utilizado para execução dos testes funcionais no Magistro foi o GEB [6]. GEB utiliza o Spock Framework[7] para realizar os testes. O Spock ao invés de usar funções para realizar os teste como é feito em teste unitários ele nos incentiva a usar frases que possam ser compreendidas por programadores ou não.

[5] <https://developers.google.com/chart/>
[6] <https://grails.org/plugin/geb>
[7] <http://spockframework.github.io/spock/docs/1.0/index.html>

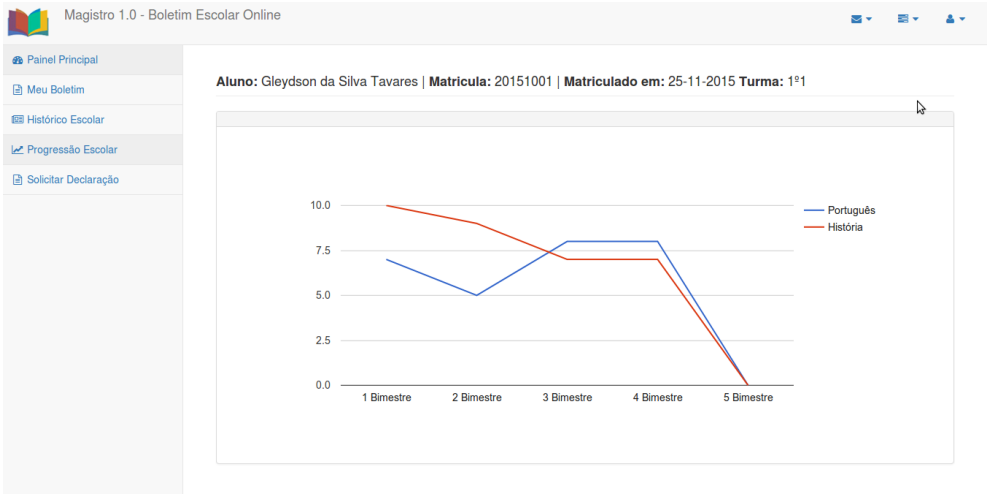


Figura 9: Desempenho anual do aluno

Para utilizar o GEB o processo foi o mesmo para todos os plugins do Grails, apenas inserir no arquivo *BuildConfig.groovy* a linha para o plugin: *compile ":geb:0.12.2"* e inserir na pasta */test/functional* o arquivo de teste.

O código 3.9 exibe o teste funcional para o CRUD de Aluno.

Código 3.9: Teste funcional para o CRUD de Aluno

```
@Stepwise
class AlunoCRUDFunc extends GebReportingSpec {
    def "nao existem alunos"() {
        when:
        to ListPage
        then:
        alunoRows.size() == 0
    }
    def "adicionar aluno"() {
        when:
        newAlunoButton.click()
        then:
        at CreatePage
    }
    def "cadastrar aluno"() {
            when:
            nome = "Gleydson"
            cpf = "412.312.285-97"
            createButton.click()
            then:
```

```
            at ShowPage
    }
}
```

A figura 3.3.6 exibe a saída do terminal para o teste funcional do CRUD do aluno. O teste é rodado pelo terminal, então o Grails abre um navegador web e roda todos os testes funcionais, caso algum erro seja encontrado uma mensagem é exibida junto com o caminho para o log de erros.

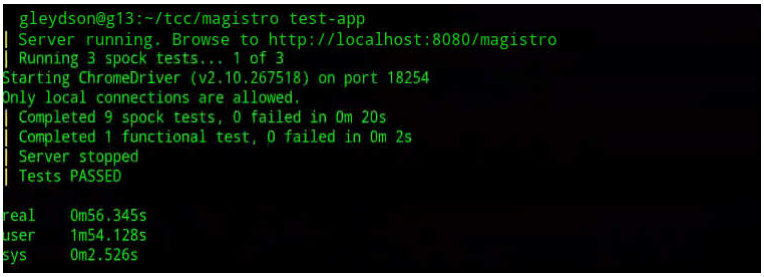


Figura 10: Saída do teste funcional

4 Considerações finais

Com a base adquirida no curso de Sistemas para Internet, nos livros, na elaboração desse relatório final de estágio e no desenvolvimento do sistema Magistro, considero todas essas coisas de grande valia para concretizar esse estágio.

O sistema Magistro permite o lançamento de notas por professores e o acesso dessas notas pelos alunos e seus responsáveis, e a emissão de alguns documentos e relatórios que dependem dessas notas. Até o momento o sistema está em fase de testes e vem cumprindo com as expectativas da direção e dos professores, porém, só estará disponível integralmente para a comunidade escolar a partir do próximo ano letivo por questões do orçamento anual e de adaptação com o novo sistema.

A principal dificuldade encontrada foi em relação ao levantamento de requisitos e de gerenciar o processo de desenvolvimento, mas essas dificuldade foram supridas com o aprendizado adquiridos nas cadeiras de gerência de projetos de software e desenvolvimento e execução de processos de software sendo essas para mim as mais importantes durante o estágio.

Um desafio futuro proposto pela direção da escola é utilizar o sistema para gerenciar atividades da secretaria escolar, como a criação da caderneta do professor online e a emissão automática da ficha do aluno o que é impossível no momento por questões burocráticas.

Referências

ANSELMO, F. *Em busca do Grails*. 1. ed. São Paulo: Visual Books, 2010. Citado na página 18.

FERREIRA, D. et al. *Um Modelo Ágil para Gestão de Projectos de Software*. 2005. Disponível em: <http://paginas.fe.up.pt/~aaguiar/es/artigos%20finais/es_final_19.pdf>. Acesso em: 03 nov 2015. Citado na página 15.

FRANCO, R. S. T. *ESTUDO COMPARATIVO ENTRE FRAMEWORKS JAVA PARA DESENVOLVIMENTO DE APLICAÇÕES WEB: JSF 2.0, GRAILS E SPRING WEB MVC*. 2011. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/492/1/CT_JAVA_VI_2010_16.PDF>. Acesso em: 03 nov 2015. Citado na página 18.

FURLAN, J. D. *Modelagem de objetos através de UML*. 1. ed. São Paulo: Makron Books, 1998. Citado na página 24.

GOODMAN, D. *JavaScript Bible*. [S.l.]: Ed Gold, 2001. 40-53 p. Citado na página 17.

GUEDES, G. T. A. *UML 2, Guia prático*. [S.l.]: Novatec, 2014. Citado na página 21.

HENSGEN, P. *Manual do Umbrello UML Modeller*. 2015. Disponível em: <https://docs.kde.org/trunk4/pt_BR/kdesdk/umbrello/uml-basics.html>. Acesso em: 03 nov 2015. Citado na página 21.

KOENIG, D. *Groovy in Action*. 2. ed. [S.l.]: Meap, 2011. 4 p. Citado na página 20.

MUNZLINGER, E. *JavaScript, Histórico e características*. 2011. Disponível em: <http://www.elizabete.com.br/site/Outros/Entradas/2011/2/20_Tecnologia_Web_files/01-JS-HistoricoCaracteristicas.pdf>. Acesso em: 03 nov 2015. Citado na página 17.

SILVA, A. M. R. da; VIDEIRA, C. A. E. *UML, Metodologias e Ferramentas CASE*. Porto: MCentro Atlântico, 2001. Citado na página 21.

SILVA, M. S. *jQuery - A Biblioteca do Programador JavaScript*. 1. ed. São Paulo: Novatec, 2008. 432 p. Citado na página 17.

WEISSMANN, H. L. *Falando de Grails*. 1. ed. [S.l.]: Casa do Código, 2015. 301 p. Citado 2 vezes nas páginas 30 e 31.