Instituto de Matemática e Estatística
Universidade de São Paulo, Brasil

Academic Devoir

André Satoshi
António Castro
Bruno Padilha
Gustavo Coelho
Luciana Kayo
Susanna Rezende
Suzana Santos
Vinicius Rezende
Wallace Almeida

Professor: Marco Gerosa
Disciplina: MAC0332 Engenharia de Software Versão: 2.0

24 de outubro de 2011

Sumário

1 Analista

1.1 Visão

"Academic Devoir" é um sistema a ser desenvolvido para uso em plataforma Web, utilizando Java como linguagem básica de desenvolvimento, e adotando como tecnologias auxiliares os frameworks vRaptor e Hibernate.

O sofware tem como característica principal a gestão de tarefas acadêmicas das turmas de variadas disciplinas, baseadas em listas de exercícios, por sua vez constituídas de questões em diversas categorias, como dissertativa e "Verdadeiro ou Falso", e de correção automática.

O uso do sistema será destinado a professores e alunos. Os primeiros podem elaborar as atividades e visualizar a correção automática efetuada. Já os últimos poderão exibir as atividades disponíveis e solucioná-las.

1.2 Requerimentos do Sistema (System-Wide Requirements)

Por ser um software de aplicação acadêmica que permitirá o acesso de vários usuários diferentes, tanto alunos como professores, o "Academic Devoir" visa uma interface simples e prática de comunicação com o usuário. Sendo assim, uma "linkagem" bem estruturada entre as páginas, bem como a clareza das informações apresentadas em cada tela, se faz imensamente necessária.

Além disso, por ser um sistema acessado por usuários de tipos diferentes, o controle de login e de permissões para a realização de tarefas, ou seja, o monitoramento do que cada tipo de usuário pode fazer, e das páginas que pode acessar, é fundamental.

Dada a proposta do sistema, de administrar turmas e disciplinas diferentes, e vários alunos, a cada semestre, e a dinânima das ações realizadas (a constante adição de novos recursos, como exercícios, e de novos usuários; a atualização de informações, como notas etc), é de extrema importância que o software tenha interfaces que agilizem certas funcionalidades, como telas para alteração de dados, e que todas as funcionalidades estejam devidamente testadas, para não causarem erros devido às constantes mudanças.

Também como pedido pelo cliente (que aqui vem a ser o próprio professor de nossa disciplina), o "Academic Devoir" deve apresentar a opção de correção automática de listas, com opções, quanto a prazos e formas de entrega, que possam ser configurados pelo professor, segundo seus próprios critérios.

1.3 Casos de Uso

As funcionalidades mostradas pelo sistema variam de acordo com o tipo de usuário, aluno ou professor, que o estiver usando. Podemos, assim, separar tais funcionalidades em casos de uso, simplificando a visão das ações permitidas pelo software.

Um caso de uso representa simplesmente uma funcionalidade dentro do sistema. O projeto acompanha um diagrama, de nome `CasosDeUso.png`, com todas as ações possíveis para os atores do software. No "Academic Devoir" o caso de uso mais geral é o login. Este é efetuado por usuários em geral, independente do tipo, resultado da herança existente entre as classes Usuário, Aluno e Professor. No diagrama, a relação de herança entre os atores é representada com uma seta direcionada ao Usuário, indicando que tanto Aluno quanto Professor apresentam a funcionalidade de login. Os usos específicos dos atores Aluno e Professor estão relacionados abaixo.

Ator: Aluno

- Cadastrar-se: um aluno pode adicionar-se ao sistema, tendo seu acesso mais restrito em relação ao usuário do tipo professor;
- Alterar dados pessoais: permite a alteração ou correção de dados após efetuado o cadastro;
- Listar turmas: essa funcionalidade apresenta duas derivações: listagem de turmas existentes no sistema e listagem de turmas na qual o aluno está matriculado. A primeira é uma listagem geral de todas as turmas disponíveis para a disciplina que o aluno vai cursar, permitindo a ele se cadastrar numa específica. A segunda é a visualização de todas as turmas na qual o aluno efetivamente se matriculou, permitindo o acesso a outros recursos relacionados a essas turmas;
- Matricular-se numa turma: como extensão do primeiro tipo de listagem, permite que o aluno escolha uma turma específica relacionada à disciplina que pretende cursar e increver-se nela;
- Visualização da turma: possível apenas após a matrícula, quando são liberados recursos como listas de exercícios;
- Visualização de listas de exercícios: dada uma turma específica, um aluno devidamente matriculado deve poder ver as tarefas disponíveis para entrega, bem como seus prazos;
- Resolução de lista: uma vez visualizada uma lista, o aluno pode decidir se entrega ou não a lista, e quando a entrega (considerando, obviamente, o prazo disponível para tal tarefa);
- Visualização de nota: o aluno tem acesso às notas das tarefas que entregou;

Ator: Professor

- Cadastrar-se: ao professor é permitido adicionar-se ao sistema com maiores privilégios de acesso do que um aluno;
- Alterar dados pessoais: permite a alteração ou correção de dado após efetuado o cadastro, da mesma forma que é permitido a um aluno;
- Gerenciamento de alunos: dadas as permissões oferecidas a esse tipo de usuário, a tarefa de gerenciamento de alunos, que foca no controle dos alunos de uma turma comandada por um professor, pode ser extendida em subtarefas;
- Listagem de alunos: subtarefa diretamente ligada ao gerenciamento, é necessário que o professor esteja na página de gerenciamento, ou que efetue previamente a listagem de turmas, escolhendo uma de suas turmas, para ter acesso aos alunos de uma turma em específico;
- Remoção de aluno: efetuada uma listagem, é possível que o professor exclua um aluno de sua turma que está presente na lista. O professor pode precisar dessa funcionalidade para o caso, por exemplo, de um aluno ter comportamento inadequado diante de algum fórum de uma disciplina, e o professor necessitar tomar providências;
- Cadastrar monitor: após a listagem, o professor poe identificar um aluno específico, e alterar seus privilégios transformando-o em monitor. Importante observar que o privilégio de um aluno como monitor muda na turma específica que ele auxilia. Para outras turmas, ele ainda é apenas um aluno comum;
- Cadastra disciplina: usuários do tipo professor tem permissão para adicionar novas disciplinas ao sistema. Efetuado o cadastro de uma disciplina, tanto o professor que a cadastrou, quanto outros professores, estão aptos a adicionar às disciplinas novas turmas;
- Cadastra turma: dadas as disciplinas disponíveis, ele pode criar uma nova turma, sendo automaticamente colocado como professor responsável por ela. Uma vez cadastrada uma turma por um professor, outro professor não pode alterá-la.
- Listar disciplinas: listagem de disciplinas nas quais o professor tem ao menos uma turma pela qual é responsável;
- Listar turmas: apresenta a listagem de turmas cadastradas por aquele professor, ou seja, as turmas pelas quais ele é responsável. Pode ser vista como uma extensão da listagem de disciplinas, uma vez que ao acessar uma disciplina o professor, consequentemente, tem acesso às turmas que comanda dentro dela;
- Cadastra lista de exercícios: ao professor é permitido criar novas listas de exercícios nas turmas para as quais leciona;

- Corrige lista: cadastrada uma lista, o professor corrige as listas resolvidas pelos alunos baseado nas respostas corretas de cada questão;
- Visualiza listas de exercícios: listagem das listas que o professor cadastrou para sua turma;
- Reordena questões: o professor pode mudar a ordem das questões que vão aparecer na lista, uma vez que a lista foi salva no banco de dados do sistema;
- Cadastro de questões: é a forma mais genérica da ação de adicionar uma nova questão que ficará disponível no banco de dados. Esse cadastro é especificado em quatro subtipos, de acordo com a espécie da questão desejada: cadastro de questão de texto, cadastro de questão de V ou F, cadastro de questão de múltipla escolha e, por fim, cadastro de questão de submissão de arquivo.

1.4 Modelo de Casos de Uso

Diagrama de casos de uso

O diagrama de casos de uso se encontra em `CasosDeUso.png` no diretório `doc/diagramas`.

1.5 Glossário

G

Gerenciar: Entende-se que um usuário está apto a gerenciar uma dada estrutura do sistema quando possui permissões e interfaces para inclusão, edição, visualização e remoção de elementos dessa estrutura.

L

Lista de exercícios: Conjunto de questões, criado para uma turma, por um professor específico responsável por tal turma. Devem ser resolvidas e entregues pelos alunos daquela turma.

Q

Questão: Entidade que representa um exercício, contendo enunciado e resposta correta. É criada por usuários do tipo professor.

T

Turma: Conjunto composto por alunos (e monitor) que cursam uma disciplina determinada oferecida por um professor em um dado semestre. Podem existir mais de uma turma, com professores distintos, para uma mesma disciplina e semestre.

U

Usuário: Aquele que interage e utiliza o sistema. Nas especificações atuais, pode ser um professor ou um aluno.

2 Gerente

2.1 Plano da Iteração

Objetivos da iteração

Após as 2 primeiras iterações, que nos deram um esboço final de como será o sistema e que implementaram parte considerável de suas funcionalidades, o objetivo agora é a melhoria de tais funcionalidades, bem como a implementação do fator diferencial do sistema, que é o seu sub-sistema de cadastro e resolução de questões e listas. Os objetivos propostos para a atual fase de desenvolvimento são:

- Complementar e aprimorar as funcionalidades elaboradas durante as duas iterações anteriores;
- Realizar o cadastro de questões e listas, bem como sua resolução;
- Melhorar a navegabilidade do sistema para fins de depuração;
- Acrescentar mais informações a documentação do projeto.

Apesar de parte dos objetivos desta iteração, que visam fazer o sistema vir a ser completo, não se faz necessário, pelo menos temporariamente, um sistema facilmente navegável e agradável visualmente. Por enquanto, apenas o mínimo necessário de tais partes foi implementado. Isso será refinado apenas quando o sistema for tido como completo do ponto de vista funcional.

Atribuições de trabalho

A cada iteração, os cargos e responsabilidades devem ser trocados. As tarefas foram atribuídas conforme o papel dos integrantes. Dependendo da dificuldade da tarefa e de sua relevância para o desenvolvimento do projeto, todos os integrantes devem contribuir.

Segue uma lista dos participantes e de seus respectivos itens de trabalho, para essa terceira iteração:

- André Satoshi Fujii de Siqueira (Arquiteto)
  - Modelagem das classes do sistema.
  - Refinamento do código do sistema.
  - Implementação do cadastro e resolução de questões e listas.

- Antonio Junior (Desenvolvedor)
  - Refinamento do código do sistema.
  - Implementação do cadastro e resolução de questões e listas.
- Gustavo Coelho (Gerente)
  - Monitoramento de algumas atividades realizadas pelos integrantes do grupo.
  - Levantamento da visão do projeto e dos requisitos cumpridos até o momento.
  - Implementação de connection pooling.
- Luciana Kayo (Analista de Requisitos)
  - Levantamento de requisitos do sistema.
  - Manutenção dos arquivos jsp.
- Susanna Rezende (Arquiteto)
  - Refinamento do código do sistema.
  - Modelagem das classes do sistema.
- Suzana de S. Santos (Documentador)
  - Estudar documentação OpenUP e instruir os integrantes do grupo sobre como deve ser a documentação.
  - Acompanhar a documentação do projeto e dar as orientações necessárias.
  - Refinamento do código do sistema.
- Vinicius Rezende (Analista de Qualidade)
  - Mapeamento ORM no hibernate.

  - Implementação de testes para fins de depuração e manutenção.

- Wallace Faveron de Almeida (Desenvolvedor)
  - Refinamento do código do sistema.
  - Criação de renderizadores para as diferentes questões.

Para maior detalhamento dos itens de trabalho, visite a página: `https://www.pivotaltracker.com/projects/355453#`

Critérios de avaliação

Para avaliarmos o desenvolvimento, usaremos os testes de Unidade e a avaliação do cliente sobre o sistema. Veja maiores detalhes sobre os testes na documentação do analista de qualidade.

2.2 Plano do Projeto

Organização

O trabalho no projeto é dividido nas seguintes áreas:

Identificação	Responsabilidades	Stakeholders
Gerente do Projeto	Atribuições de caráter decisório e estratégico quanto aos rumos do projeto.	Gustavo
Analistas	Definir e aprovar os requisitos e especificações de negócio do sistema.	Luciana
Arquiteto do Projeto	Definir a arquitetura a ser utilizada no sistema.	André e Susanna
Documentação	Documentar	Suzana Colaboradores: Todos
Programadores	Implementar o sistema conforme as especificações.	Wallace e Antònio Colaboradores: Todos
Testes	Padronizar os testes, homologar.	Vinícius

Medições

A cada item de trabalho atribuímos pontos, que representam horas de dedicação (1 ponto equivale a uma hora). Na página `https://www.pivotaltracker.com/projects/355453#`, temos os seguintes agrupamentos de itens de trabalho:

1. Current (trabalho em desenvolvimento, concluído ou a ser desenvolvido em breve)
2. Backlog (trabalho a ser desenvolvido em breve)
3. Icebox (trabalho a ser desenvolvido sem data para início)

Objetivos

Criar um sistema de resolução e correção de listas de exercício na Web. O desenvolvimento consistirá de 3 iterações (cada uma com 3 semanas de duração) durante as quais serão implementadas as histórias:

- Aluno se cadastra
- Login de aluno/professor

- Professor gerencia alunos (CRUD)
- Professor cadastra disciplina
- Professor cadastra turma
- Professor cadastra lista de exercício
- Professor cadastra questão de texto
- Professor cadastra questão de V ou F
- Professor cadastra questão de múltipla escolha
- Professor cadastra questão de submissão de arquivo
- Aluno se matricula em uma turma de uma disciplina
- Professor lista respostas de questão de texto que ainda não foram corrigidas e pode corrigi-la
- Professor pode reordenar questões de uma lista
- Aluno entra em uma disciplina e visualiza as listas a serem feitas e as já corrigidas
- Aluno resolve uma lista de exercícios
- Professor determina o tipo de matricula em uma turma (imediato ou com prazo definido)

Aguardamos mais informações do professor para completar a lista de histórias a serem implementadas.

2.3 Lista de Riscos

Perguntar para o professor o que seriam os riscos do projeto.

2.4 Avaliação de status

Aguardando feedback do cliente.

2.5 Lista de Itens de Trabalhos

A lista dos itens de trabalho pode ser visualizada em: `https://www.pivotaltracker.com/projects/355453#`

3 Arquiteto

3.1 Caderno de Arquitetura

Objetivos arquiteturais

O planejamento da arquitetura do sistema tem por objetivos minimizar riscos, organizar o desenvolvimento e dar uma visão geral do projeto para todos os integrantes do grupo, além de abrir um canal de comunicação para discutir dificuldades que surgem na implementação. Isso possibilita a criação de um sistema robusto para manutenções a longo prazo.

O caderno de arquitetura descreve o contexto para o desenvolvimento de software, contendo as decisões, a análise racional, as explicações e as implicações em formar a arquitetura do sistema. Seu propósito é descrever o contexto e a perspectiva do sistema para garantir a integridade e compreensão.

Esboço do diagrama de classes

Obs.: Os nomes dados aos métodos e atributos nesse esboço são apenas descritivos. Na implementação, podemos ter nomes diferentes, seguindo os padrões do código.

```
Classe abstrata

Classe abstrata
Usuário {
    ----- Atributos -----
    id
    nome
    login
    senha
    email
    privilégio
    ----- Métodos -----
    fazerLogin()
    alterarDados()
}

Classes que implementam Usuário:
    Aluno {
        ----- Atributos -----
        lista de respostas
        lista de turmas (ids) [agregação n - n]
        ----- Métodos -----
        fazerCadastro()
        listarTurmas()
        inscriçãoNaTurma()
        remoçãoDaTurma()
        entregarLista()
    }

    Professor {
        ----- Atributos -----
        lista de turmas (ids) [agregação 1 - n]
        ----- Métodos -----
        criarTurma()
        removerTurma()
        cadastrarDsiciplina()
        cadastrarLista()
        removerLista()
        cadastrarQuestão()
```

```
        cadastrarAlunoNaTurma()
        removerAlunoDaTurma()
        cadastrarOutroProfessor()
    }


Turma {
    ----- Atributos -----
    id
    disciplina
    tipoDeMatricula
    períodoDeMatrícula
    professor responsável
    lista de alunos (ids)
    lista de atividades (lista de lista de exercicíos) [agregação 1 - n]
    ----- Métodos -----
    listarAlunos()
    listarListasDeAtividades()
}

Disciplina {
    ----- Atributos -----
    id
    nome
    lista de turmas [composição 1 - n]
}

ListaDeExercicios {
    ----- Atributos -----
    id
    propriedades
    prazoDeEntrega
    visibilidade
    lista de questões (ids) [agregação n - n]
    lista de listas de respostas (lista de objetos do tipo ListaDeRepostas) [composição
    lista de turmas
}

ListaDeRespostas {
    ----- Atributos -----
    id
    aluno
    propriedades (estado)
    lista de exercícios
    lista de respostas [composição 1 { n]
```

```
    lista de notas
    nota final
    ----- Métodos -----
    notaDaLista()
    salvaLista()
}


Classe abstrata
Resposta {
    ----- Métodos -----
    salvarRespota()
    alterarResposta()
    removerResposta()
}

Classes que implementam Resposta:
    Multipla escolha {
        ----- Atributos -----
        alternativa escolhida pelo aluno
    }

    VouF {
        ----- Atributos -----
        alternativa escolhida pelo aluno
    }

    Submissão de arquivo: {
        ----- Atributos -----
        arquivo enviado pelo aluno
    }

    Texto {
        ----- Atributos -----
        texto da resposta dada pelo aluno
    }


Classe abstrata
Questão {
    ----- Atributos -----
    id
    enunciado
    tags predefinidas
    tags definidas pelo usuário
```

```
    ----- Métodos -----
    renderização //abstrato
    devolve tipo //abstrato
}

Classes que inplementam Questões:
    Multipla escolha {
        ----- Atributos -----
        resposta
        ----- Métodos -----
        devolve alternativas
        renderização
        devolve tipo
    }

    V ou F {
        ----- Atributos -----
        resposta
        ----- Métodos -----
        devolve resposta
        renderização
        devolve tipo
    }

    Submissão de arquivo: {
        ----- Métodos -----
        devolve tipo
    }

    Texto {
        ----- Atributos -----
        resposta correta
        ----- Métodos -----
        devolve alternativas
        renderização
        devolve tipo
    }


QuestaoDaLista: {
    ----- Atributos -----
    questão
    valor
ordem
}
```

```
BancoDeQuestões {
    lista de questões
}
```

Comentários sobre as decisões tomadas

Aqui juntamos alguns dos comentários, discussões e problemas em aberto mais relevantes com relação às decisões de arquitetura tomadas no projeto. A maioria desses comentários surgiram espontaneamente das discussões entre os membros da equipe.

Classes que implementam usuário (Aluno e Professor):
Como Aluno e Professor compartilham muitos campos, optamos por essa herança, evitando assim repetição de código.

Atributo lista de turmas (ids) em Aluno e Professor:
Esse campo não está em "Usuário", pois a relação ente Aluno e Turma é n-n, enquanto a relação entre Professor e Turma é 1-n.

Método entregar listas na classe aluno:
O aluno também pode salvar uma lista além de enviá-la (ou finalizá-la), dependendo das configurações da lista de exercícios.

Atributo períodoDeMatrícula em turma:
Dependendo do tipo de matrícula, haverá um período determinado para o aluno se matricular em uma turma.

Método listarListasDeAtividades em turma:
Listar as listas de exercícios.

Atributo prazoDeEntrega da listaDeExercícios:
O professor pode definir um prazo de entrega.
Dúvida: Colocar horário limite também?

Atributo visibilidade da listaDeExercícios:
Dita se a lista é visível ou invisível para os alunos.
Dúvida: O professor pode programar uma data para a lista passar a ficar visível?

Atributo lista de listas de respostas da listaDeExercícios:
Lista com as listas de exercícios respondidas pelos alunos (exercícios esses que estão na mesma ordem em que aparecem na lista de questões).

Atributo "id da listaDeExercícios" na listaDeRespostas:
Pergunta: Dado que a ListaDeExercicios tem um id, não seria interessante que a lista de respostas carregasse também o id da lista a qual ela pertence? Ou isso seria redundante?
Resposta: A ListaDeRespostas é sempre acessada a partir da ListaDeExercícios, certo? Nesse caso, talvez não haja necessidade de ListaDeResposta guardar o id da ListaDeExercícios.

Atributo estado em propriedades da listaDeRespostas:

A lista pode não ter sido inicializada, ter sido inicializada ou finalizada.

Atributo lista de respostas na listaDeRespostas:
Respostas dadas pelos alunos, de fato.

Atributo corrigido na classe Resposta:
Pergunta: Precisamos desse status?
Resposta: Achamos que não.

Classes que implementam Resposta:
A princípio, pensamos nessas especializações de Resposta. Há também a possibilidade de implementar todas as respostas em uma mesma classe. Acabamos escolhendo essa última opção.

Atributo alternativa escolhida pelo aluno na classe Resposta:
A princípio pensamos em permitir várias alternativas corretas, mas finalmente decidimos que uma questão de múltipla escolha permitiria apenas uma alternativa correta.

Classe abstrata Questão:
A herança entre questão e os variados tipos de questão permite o polimorfismo e o aproveitamento de alguns campos entre os tipos de questões.

Classe BancoDeQuestões:
Decidimos por criar um "banco de questões" onde os professores poderiam compartilhar questões entre si. Para isso, a classe Questão não podia conter nenhum atributo que fosse específico da instância dessa questão em uma lista específica (como valor e ordem). Para resolver esse problema criamos uma classe chamada QuestãoDaLista. Cada objeto dessa classe contém uma questão, seu valor e sua posição na lista.

Classe QuestãoDaLista:
Classe para guardar o valor da questão e a questão.

Atributo valor na classe QuestãoDaLista:
O professor pode atribuir um valor para cada questão.
A princípio essa propriedade estava na classe Questão, mas como depende do contexto em que a questão está sendo utilizada, colocamos nessa classe.

Atributo questão da classe QuestãoDaLista:
Dúvida: Aqui seria a questão mesmo ou Id?

Discussão:

- Se fosse a questão mesmo, elas não poderiam ter valores diferentes em listas de exercícios diferentes (que podem ser dadas por professores distintos). Sendo apenas a id não teríamos esse problema, certo?

- Pode ser a questão sim, já que agora o valor é um atributo da QuestaoDaLista. A diferença é que 2 listas não poderão conter a mesma QuestaoDaLista.

Diagramas de classe

Os diagramas de classe se encontram nos seguintes arquivos (no diretório `doc/diagramas`):

- `Dependência.png`: mostra as dependências entre as classes
- `Associação.png`: mostra as associações entre as classes
- `Herança.png`: mostras as heranças de classes

4 Desenvolvedor

4.1 Design

Design structure

Subsystems

Patterns

Overview
Structure
Behavior
Example

Requirement realizations

View of participants

5 Testador

A maioria dos testes realizados até o momento são testes nas classes Controller, ou seja, que testam se os Controllers chamam os métodos apropriados dos DAOs e se o usuário é redirecionado para as páginas certas.

Há também testes que consistem em verificar se os diferentes objetos (Alunos, Professores, etc...) estão sendo corretamente inseridos em suas tabelas no banco de dados.

Para não afetar o banco de dados, todas os objetos que lidam com o banco de dados diratemente são mocks (imitações) dos objetos reais.

5.1 Test Case

Test Case ID - Test Case Name:
Descrição:
Pré-condições:
Pós-condições:
Dados requeridos:

5.2 Test Log

Produzido automaticamente.