CENTRO UNIVERSITÁRIO UNIFACVEST

CURSO DE CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

LEONARDO BOSQUETT

Unifacvest: Reestruturação do site

LAGES,

2014

LEONARDO BOSQUETT

Unifacvest: Reestruturação do site

Projeto apresentado à banca examinadora de trabalhos do curso de Ciência da computação para análise e aprovação.

LAGES,

2014

LEONARDO BOSQUETT

Unifacvest: Reestruturação do site

Trabalho de conclusão do Curso de Ciência da Computação apresentado ao Centro Universitário FACVEST como parte dos requisitos para obtenção do título de bacharel em Ciência da Computação.

Prof. MSc. Márcio José Sembay.

Lages, SC ____/____/2014. Nota_____ _____

_____

_____

_____

LAGES,

2014

RESUMO

O presente trabalho visa buscar a restruturação do sistema web da instituição Unifacvest a fim de melhorar o serviço aos alunos, professores e público em geral. São definidos também os seguintes objetivos: Criação de uma nova interface do usuário; Inserção de novas funcionalidades; Socialização; Tornar o sistema acessível aos dispositivos móveis. A metodologia de pesquisa se baseia no conhecimento científico e é do tipo exploratória, a fim de descrever o que é cada item envolvido no projeto. Será utilizado o padrão MVC para desenvolvimento. Este projeto foi criado para resolver o problema atual de design do site da instituição, já que o mesmo com o passar do tempo precisa de atualizações.

Palavras-chave: Sistema Web, MVC, dispositivos móveis.

ABSTRACT

This paper aims at seeking to restructure the institution Unifacvest web system in order to improve service to students, teachers and the general public. The following objectives are also defined: Creating a new user interface; Insertion of new features; socialization; Make the system accessible to mobile devices. The research methodology is based on scientific knowledge and is the exploratory type, in order to describe what each item involved in the project. The MVC pattern will be used for development. This project was created to solve the current problem of other website design, since even with the passage of time needs an update.

Key-words: Web system, MVC, mobile.

RESUMEN

El trabajo tiene como objetivo tratar de reestructurar la institución sistema web Unifacvest con el fin de mejorar el servicio a los estudiantes, profesores y público en general. Los siguientes objetivos se definen también: Creación de una nueva interfaz de usuario; La inserción de las nuevas características; Socialización; Hacer que el sistema de acceso a los dispositivos móviles. La metodología de investigación se basa en el conocimiento científico y es el tipo exploratorio, con el fin de describir lo que cada elemento involucrado en el proyecto. El patrón MVC se utilizará para el desarrollo. Este proyecto fue creado para solucionar el problema actual de otro diseño de sitios web, ya que incluso con el paso del tiempo necesita una actualización.

Palabras clave: Sistema web, MVC, móvil.

LISTA DE SIGLAS

ABNT - Associação Brasileira de Normas Técnicas

ASP - *Active Server Pages*

B/M - *Bundling* e Minificação

CSS - *Cascading Style Sheets*

HTML - *Hypertext Markup Language*

HTTP - *Hypertext Transfer Protocol*

IIS - *Internet Information Services*

ISO - *International Organization for Standardization*

KB - *Kilobytes*

LINQ - *Language Integrated Query*

MS - Milissegundo

MVC - *Model View Controller*

RFC - *Request For Comments*

SBC - Sociedade Brasileira de Computação

SQL - *Structured Query Language*

W3C - *World Wide Web Consortium*

XHR - *XmlHttpRequest*

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

I. INTRODUÇÃO

O Design não é um padrão definido por uma norma, mas sim uma aceitação vista pela sociedade dependendo de sua cultura e visão. O modelo atual da universidade já não é mais aceito e por si só possui modelos complexos que se enquadram facilmente no padrão MVC.

> A implementação do MVC com o ASP.NET começa com um simples sistema, escrito uma única página, com a lógica da aplicação embutida nos modelos de apresentação. Com o crescimento da complexidade, o código por trás feito em ASP.NET é usado para separar a parte de apresentação (View) do código modelo/controlador. Isso funciona bem, desde que os requerimentos iniciais levem à reutilização de código e que tenha controladoras para evitar a redundância na aplicação. Até este ponto modelos independentes são criados para abstrair a lógica de negócio e o código é utilizado para adaptar o modelo a uma View. A implementação então finaliza com uma discussão sobre os testes desta aproximação do MVC. A utilização do MVC é focada no modelo e na View. A Controller está em uma posição menor. De fato, a Controller que trabalha neste modelo é realmente uma classe Controller no ASP.NET, ela é responsável por manipular os eventos do cliente (requisições e ações) e instruir a aplicação a responder adequadamente, neste caso os eventos está por trás do código da página. Em aplicações Web dinâmicas, muitas tarefas comuns são repetidas durante a requisição de uma página, como autenticação, validação, extração de parâmetros e apresentação dos modelos relacionais do banco. Não sendo gerenciáveis, estas tarefas são rapidamente duplicadas no código desnecessariamente por que estas dependem da requisição do usuário e de uma resposta apropriada do sistema, o lugar ideal para se colocar este comportamento é em uma Controller. MICROSOFT CORPORATION (2014 p. 01).

O presente trabalho atualiza o site para os padrões atuais de projeto, design e visual mais aceitos bem como a inclusão de recursos que facilitem a utilização dos alunos e professores, para isto foram utilizadas ferramentas de ponta dos maiores profissionais da web bem como alguns de seus conselhos, esse projeto não tinha pretensão de criar uma revolução no site e está sujeito a uma nova atualização em um curto período de tempo devido à rápida transição de novos padrões.

I.1 Justificativa

O sistema atual da instituição apresenta não um defeito, mas uma queixa de design devido ao tempo disponibilizado online, muitos dos alunos e professores se queixam da sua interface e utilidade como é visto na figura 1.

> O site não segue atualizações internacionais (padronização), o que demonstra incompatibilidade com plataformas Mobile. A atualização desses padrões permitiria uma melhor visualização e design especifico. A compatibilidade e ergonomia promoverão melhor entendimento pelo usuário, atingindo uma meta de usabilidade mais coerente. (SEMBAY, 2014)

Para tal foram utilizadas técnicas de desenvolvimento que não só melhorassem o visual, mas também possibilitassem uma melhor interpretação pelos navegadores recentes, a fim de obter a correta estruturação de todas as páginas da interface.

Figura 1 – Sistema da Instituição (02/03/2014),

Fonte: Unifacvest.

Como vemos na figura 1, este é o sistema utilizado atualmente na instituição, onde o layout apresenta uma falha gráfica deixando um espaço em branco em um lugar indevido, pois este possui visualização direta.

> Os *links* funcionam bem, mas os *links* secundários das páginas estão fora do lugar, talvez pelo esquema de cores que não se misturam muito bem, do ponto de vista de *design* qualquer coisa pode ser feita nesse caso, Jake também sugere que os textos na página possuam um espaçamento e tamanho de fonte maior para que tudo não fique misturado durante a leitura. (Entrevista com ROCHELEAU **designer e pesquisador da Google**).

Seguindo os conceitos de Rocheleau, os links serão organizados, porém mantidos os redirecionamentos e com melhor estruturação visual.

1.2 Objetivos

1.2.1 Objetivo Geral

Reestruturar o sistema web da instituição Unifacvest a fim de melhorar o serviço aos alunos, professores e público em geral.

1.2.2 Objetivos específicos

Com o estudo feito no sistema atual da instituição de ensino, foi obtida uma lista de objetivos específicos baseado na necessidade primária de um serviço web:

a) Criado uma nova interface do usuário acessível de dispositivos móveis;
b) Gestão de usuários com níveis de acesso, banners, páginas institucionais, cursos, turmas, eventos e semanas acadêmicas.
c) Criado sistema de troca de mensagens entre alunos e professores;
d) Contas integradas com o Sistema Unimestre.
e) Integrado Sistema de gestão da área da saúde (projeto específico).
f) Disponibilização de ferramenta de auxílio acadêmico para professores (trabalho paralelo de Carlos Guilherme Coelho).

1.3 Estruturação do TCC

O presente trabalho se divide em oito partes, cada uma descrevendo detalhes de uma área específica a fim de melhorar a separação do conteúdo.

A primeira se refere à introdução, abrange o conteúdo geral do trabalho e uma visão do que se trata.

A segunda trata da revisão bibliográfica, as linguagens utilizadas no projeto e uma breve revisão de um sistema web.

A terceira mostra as técnicas para se melhorar a qualidade do sistema.

A quarta delimita as funções dos principais usuários do sistema, demonstrando também as limitações referentes ao próprio grupo.

A quinta descreve a utilização de redes sociais e sua delimitação e utilidade para com a instituição, seus pontos fortes e como funcionaria a troca de mensagens entre os alunos e professores fora da instituição.

A sexta esclarece o uso dos eventos do servidor e tecnologias que habilitem seu uso.

A sétima parte do trabalho fala sobre a metodologia de pesquisa do projeto, contendo o cronograma e as principais ferramentas utilizadas no desenvolvimento do projeto final.

A oitava parte mostra o Pré-projeto do sistema e como serão dividas as áreas de informação dentro do sistema.

Logo em seguida vêm as conclusões e referências utilizadas dentro do trabalho.

II. REVISÃO BIBLIOGRÁFICA

2.1 Sistemas Web

A base de um sistema web é a sua comunicação pela rede, troca de informações através da internet, assim se baseia o que um website deve mostrar ao público, informação. Para tal informação ser transmitida existe um cliente e um servidor que operam sobre o protocolo HTTP, no lado do servidor existe uma aplicação que cria as respostas feitas pelos clientes durantes as requisições, essas definições estão disponíveis e publicadas pela RFC 2616.

2.1.1 História da Internet

Segundo a RFC 2616, o protocolo atual para texto HTML (padrão de páginas web) é o HTTP 1.1. Tim Berners-Lee, participante do grupo W3C foi o inventor do HTML em 1992.

> Quando a internet estava começando, ninguém tinha ideia de quão grande ela se tornaria. As empresas de computadores não achavam que seria uma grande coisa, nem mesmo as empresas de telefone ou a cabo. De difícil acesso e operação, parecia provável que permaneceria uma ferramenta secreta do Departamento de Defesa e da Academia. Porém, a internet cresceu, e cresceu, e cresceu. Ela começou com um punhado de usuários na metade da década de 1960 e alcançou 2 bilhões em 2010. As estimativas preveem que haverá mais de 3 bilhões de usuários em 2015. Vilas na Indonésia e na Índia já tem acesso à internet antes mesmo de terem eletricidade. (BALTZAN 2012 p. 60).

Como a internet hoje está em um nível global era de se esperar que houvesse uma padronização rigorosa como a W3C seguindo das formulações feitas pelas *RFC's*. Partindo deste princípio o servidor web, tal como Apache e IIS, também seguiu estes padrões.

2.1.2 Servidor

Segundo a RFC 2616, o servidor deve responder as requisições HTTP com os códigos de resposta adequados, sendo eles:

a) 1xx: Com propósitos de informação
b) 2xx – Sucesso no processamento
c) 3xx – Redirecionamento de requisição
d) 4xx – Erro na requisição do cliente
e) 5xx – Erro no processamento interno do servidor

2.1.3 C#

A linguagem C# propriamente dita surgiu em 2001, não sofreu grandes mudanças, porém teve diversos acréscimos junto com o *Framework*. Os artigos da MSDN mantém atualizadas as modificações das novas versões do .NET disponíveis com o título "*What's new in the .NET Framework* [versão]".

> Em um curto período da história, a tecnologia Microsoft .NET se tornou rapidamente uma plataforma de programação popular para desenvolver aplicações para estações e servidores Windows. Embora muita atenção da mídia esteja focada nas capacidades das aplicações Web em .NET, existem muitas outras características que são úteis aos programadores Windows. Uma dessas características é a nova linguagem de programação C#, desenvolvida especificamente ao .NET, se tornou uma plataforma largamente utilizada por programadores que desejam criar aplicações para Windows tanto em rede quanto autônomas. A linguagem provê muitos recursos que ajudam a criar aplicações robustas para Windows. Muitos programadores estão migrando ao C#, para tirar vantagens destes recursos. (BLUM R. 2006 p. 5).

No presente projeto será utilizada a versão 4.5.1, atual e estável em 04/03/2014. Nessa versão temos melhorias na questão de organização de código, tarefas assíncronas agora serão tratadas com as palavras chave *async* e *await.*

2.1.4 Razor

De fato o Razor aumenta em muito a interpretação do que será apresentado na interface do usuário.

> O ASP.NET *Razor* é um motor que possibilita a inserção de lógica de aplicação diretamente na camada de visualização do projeto, por exemplo, é possível inserir a sintaxe *razor* junto dos códigos HTML dentro da mesma página, a sintaxe do ASP.NET *Razor* é extremamente simplificada, baseado na linguagem C# .NET, embora seja possível utilizar também *Visual Basic*. O caractere especial @ inicia uma expressão, um bloco com uma única instrução ou várias, por exemplo, podemos inserir um código Razor diretamente na marcação HTML, utilizando o caractere @ para iniciar uma expressão de condição ou apresentar o conteúdo de uma variável. (TADASHI, 2011, p. 1).

```
<!DOCTYPE html>

<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>Razor</title>
</head>
<body>
    @{ var engine = "Razor"; }
    @{ var IDE = "Visual Studio"; }

    <p>A IDE utilizada é o: @IDE </p>
    <p>A view engine utilizada é o : @engine</p>

    @{    var texto = "O Visual Studio e o Razor são tecnologias  ";
```

```
            var empresa = "Microsoft.";
            var textoCompleto = texto + " : " + empresa;
    }

    <p>@textoCompleto</p>

</body>
</html>
```

Figura 2 – Código HTML com Razor

Fonte: O próprio Autor.

A figura 2 contém um código demonstrando como integrar o HTML e a linguagem dinâmica (no caso C#). É possível assim ter uma leitura mais fácil do que é código e do que é marcação do que o antigo modo que o ASP.NET disponibilizava.

Os principais modos de integração do Razor no HTML são vistos no quadro 1.

Expressão	Significado
@	Direta, utilizada para informar valor de variáveis.
@(<expressão>)	Expressão que retornará um resultado a ser mostrado.
@{<expressão>}	Conjunto de expressões que podem ou não mostrarem resultado, mais utilizado para definição de estados e operações da View.

Quadro 1 – Expressões do Razor

Fonte: (TADASHI 2001 p. 1).

O Contexto de código após aberta a expressão é o mesmo do .NET comum. É possível, dentro de uma expressão programar normalmente utilizando os recursos e palavras chaves do C#.

2.1.5 MVC

Não para unir o código, mas sim para definir o que cada parte ficaria responsável e devolver os dados corretamente ao usuário sem perda de desempenho, sem aumento excessivo na quantidade de código e ainda sim manter cada módulo com sua responsabilidade.

> O propósito de muitos sistemas de computadores é devolver informação de um banco de dados e mostrar ao usuário. Após o usuário alterar os dados, o sistema guarda as atualizações no banco de dados. Devido o fato de a informação estar entre o banco de dados e a visualização presente do usuário, você pode ser encorajado a unir estas duas partes para reduzir a quantidade de código e aumentar a desempenho da aplicação. (FOWLER; MATRIN, 2014, p. 1).

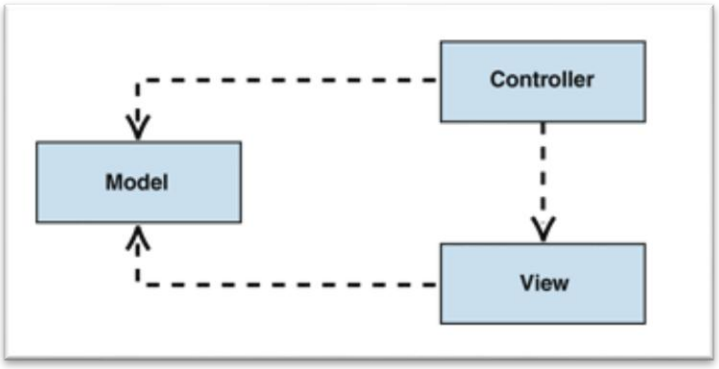

Figura 3 – Esquema do MVC

Fonte: Microsoft (2014 p. 1).

A Figura 3 mostra como o MVC possui dependências entre suas partes, A *Controller* é responsável por realizar as ações do usuário, coleta dados da *Model* e repassa a *View*, A *View* é responsável por mostrar as informações ao usuário e assim possui dependência direta com a *Model*, também pode mostrar os comandos possíveis do usuário que serão enviados de volta a *Controller*, A *Model* é responsável pela guarda da informação, lógica do sistema e regras de negócio, não possui dependências com a *Controller* ou a *View*.

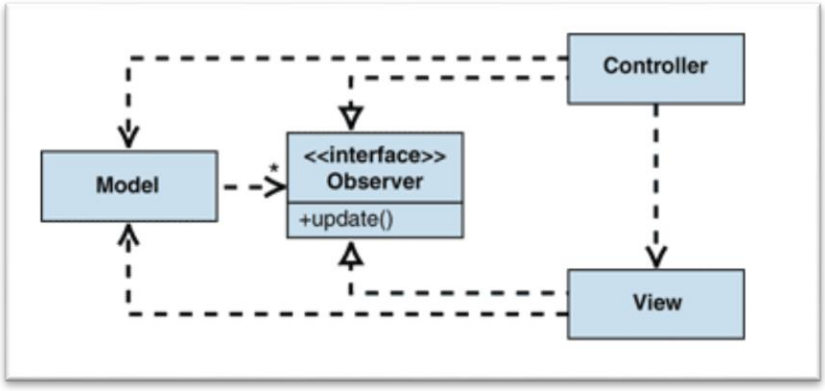

Figura 4 – Esquema de atualização dos dados da interface

Fonte: Microsoft (2014 p. 1).

A figura 4 mostra como utilizar o padrão *Observer* dentro do MVC, recurso necessário quando se precisa alterar os dados da View e notificar a *Model* sobre o evento sem injetar dependências entre as duas partes.

> Umas das motivações de utilizar MVC é tornar a *Model* independente da *View*. Se a *Model* tiver que notificar a *View* sobre mudanças, você terá que reintroduzir a dependência que esta tentando evitar. Felizmente, o padrão *Observer* provê um mecanismo para alertar outros objetos de suas mudanças de estado sem introduzir dependências nele. Cada *View* implementa a interface *Observer* e registra um modelo. O Modelo mantém uma lista de todos os observadores que estão inscritos para receber atualizações. Quando a *Model* muda, notifica todos os observadores sobre sua mudanças, um a um. Essa técnica se chama *Publish-Subscribe*. A Model nunca precisa de informações específicas da *View*. De fato, em cenários em que a Controller precisa ser informada das atualizações da *Model* (por exemplo para habilitar ou desabilitar opções do menu), todas as *Controller*s precisam implementar a interface *Observer* e se inscrever nas atualizações da *Model*. Em situações que existem muitas *Views*, faz sentido definir múltiplos objetos, cada um que descreva uma mudança específica da Model. Cada View pode se inscrever apenas nos tipos que são relevantes à *View*. (FOWLER; MATRIN, 2014, p. 1).

Esse tipo de situação ocorre bastante no uso de mensagens instantâneas, a Web principalmente costuma manter a dificuldade de atualização dos dados já que a página requisitada tende a ser estática e é preciso realizar atualizações constantes em determinadas situações. O padrão *Observer* será utilizado no projeto para envio e recebimento de mensagens entre os alunos e professores, já que não será necessário atualizar a página toda, mas apenas a frase trocada entre eles como é visualizado na figura 5.

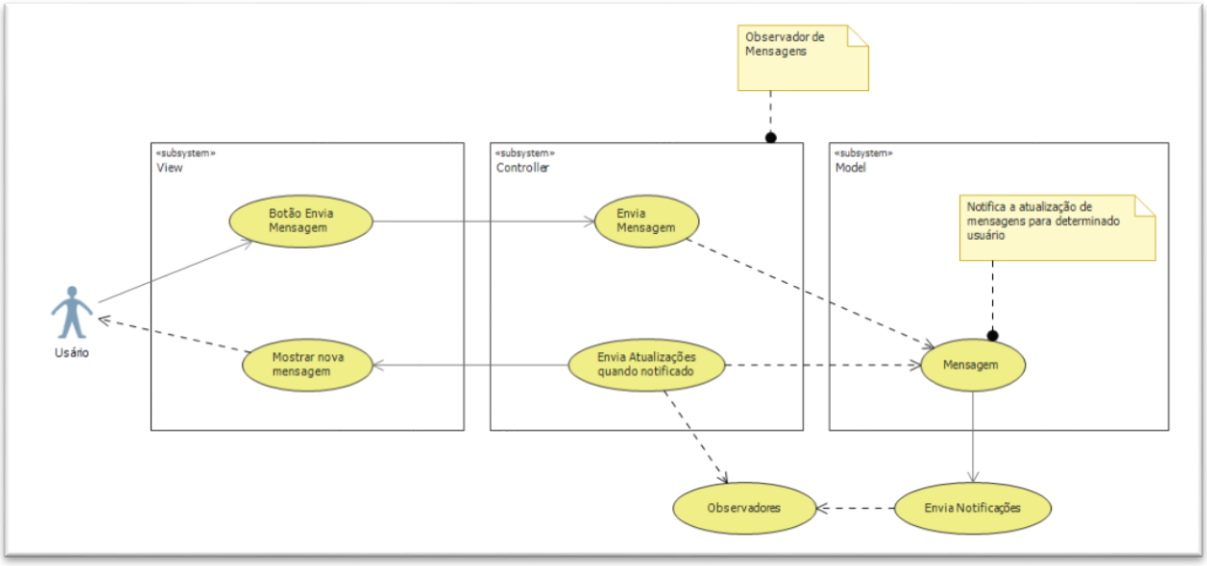

Figura 5 – Digrama de caso de uso das mensagens usando Observer

Fonte: O próprio Autor.

Este padrão precisa enviar e receber dados mesmo após a página ter sido totalmente transmitida, durante o desenvolvimento podem ser escolhidas várias opções de implementação como eventos do servidor, *web sockets* ou um monitoramento constante das mensagens (que não será escolhido, é claro por ser um padrão inadequado, com perda de desempenho e qualidade de serviço).

2.1.6 SQL

A linguagem SQL mantém sua estrutura documentada pela ISO (Organização Internacional para padronização). As regras que definem a padronização do SQL segundo a ISO podem ser vistas na figura 6:

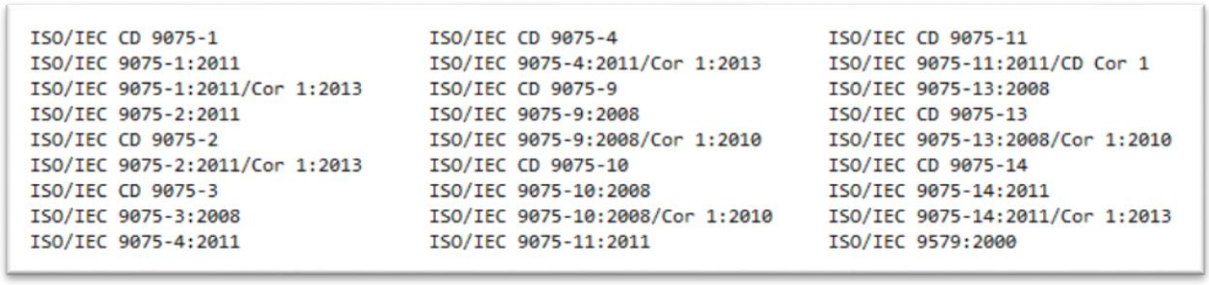

Figura 6 – Lista de regras de padronização para SQL

Fonte: O próprio Autor.

> Linguagens de programação vêm e vão constantemente, poucas linguagens usadas hoje tem seu início com mais de uma década. Alguns exemplos são *COBOL*, que ainda é utilizado em ambientes com mainframe, e C, que ainda é muito popular para sistemas operacionais, desenvolvimento de servidores e sistemas embarcados. Na arena do banco de dados, nós temos *SQL*, no qual sua criação está em torno de 1970. (BEAULIU, 2009, p. 9).

Não vamos nos ater muito neste tópico, o SQL no projeto será muito pouco utilizado devido o uso da tecnologia LINQ do .NET Framework, no qual desenvolve as tabelas e operações durante as operações internas do servidor, o que economiza muito tempo durante o desenvolvimento da aplicação. Algumas operações, no entanto podem ser úteis para visualização dos dados no banco com vista na figura 7, por exemplo.

```
select * from sys.tables;

-- consulta para retornar as tabelas dentro do banco de dados
```

Figura 7 – Consulta utilizada no SQL Server Management Studio

Fonte: O próprio Autor.

2.1.7 Cliente de um Sistema Web

2.1.7.1 HTML

Para um site, o HTML (*Hypertext Markup Language*) representa toda a parte de estruturação e visualização dos dados na página, no projeto será utilizado em enorme escala, mas somente no escopo das *Views*, dentro dos padrões da W3C, o órgão regulamentador da linguagem.

Segundo CASTRO, "A linguagem HTML iniciou no começo dos anos 90 como um pequeno documento detalhando os elementos necessários para se construir uma página Web".

2.1.7.2 HTML5

O *HTML5* introduz novas *tags* à linguagem, as quais podem definir melhor a visualização do corpo da página, como é possível visualizar no quadro 2:

audio	Conteúdo de música
video	Conteúdo de vídeo
source	Define a fonte de informação das *tags* acima.
track	Define o corte de reprodução de vídeo e de áudio.
embed	Invólucro para código embarcado, plug-ins, por exemplo.
datalist	Opções pré-definidas para dados de entrada.
keygen	Define chaves de validação para formulários
output	Resultados de operações na página
header	Define o cabeçalho da página, no contexto de visualização.
hgroup	Agrupa elementos do cabeçalho
nav	Para links de navegação no documento.
section	Define uma seção no documento.
main	Conteúdo principal da página.
article	Artigo da página.
aside	Predefinição para conteúdo que ficará ao

	lado do documento.
footer	Define o rodapé da página.
details	Define detalhes de um elemento, pode ser omitido.
summary	Define a parte visual dos detalhes
figure	Para mídia como fotos ou ilustrações.
figcaption	Texto da figura.
mark	Pra marcação de texto, destaque.
time	Para utilização de tempo, data.
bdi	Texto bidirecional, no contexto de formatação.
wbr	Localiza uma possível quebra de linha, indica a melhor posição.
dialog	Caixa de diálogo.
command	Botão de comando do usuário.
meter	Medidor.
progress	Mostra o progresso de uma lenta operação.
ruby	Anotação *Ruby* (Leste asiático).
rt	Pronunciamento de caracteres (Leste asiático).
rp	Mostra que o browser não suporta anotações em *Ruby*.

Quadro 2 – Tags HTML e seus significados.

Fonte: (W3C, 2014, p. 1).

No quadro 2 temos a listagem das *tags HTML,* as tags *HTML5* substituem diversas vezes a comumente *div,* utilizada em grande escala, algumas *tags* podem ser reproduzidas parcialmente ou totalmente no *HTML4* apenas com notação de classes em seus atributos.

O recurso é relativamente novo, mas já está documentado pela W3C e é suportado quase que totalmente nos maiores navegadores utilizados pela população.

2.1.8 Javascript

A principal utilidade desta linguagem é retirar o estado estático e inanimado das páginas web, dar vida e funcionalidade aos elementos.

Java Script é uma linguagem de programação que sobrecarrega o HTML com animação, interatividade e efeitos visuais dinâmicos. MCFARLAND, D. (2011, p. 1).

Java Script foi inventado pela Netscape em 1995... Muitos dos navegadores hoje como Firefox, Safari, Chrome, Opera e Internet Explorer 11 tem padronizado a linguagem em vários pontos tornando fácil de manuseá-la. MCFARLAND, D. (2011 p. 12).

Pela compatibilidade e, principalmente, interatividade esta linguagem será adicionada ao projeto, não sozinha, mas com o auxílio de *JQuery*.

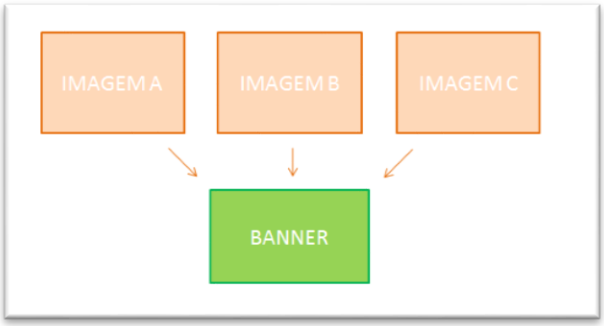

Figura 8 – Banner com várias imagens utilizando Java script e JQuery

Fonte: O próprio Autor.

Como podemos ver na figura 8, podemos criar um banner rotativo de imagens, *Java script* torna isso possível.

A linguagem é de fácil interpretação, obviamente muito de sua sintaxe deriva do Java comum, isto é observado na figura 9.

```
// declaração de funções não tem restrição de localização,
// podem estar dentro de otras funções inclusive, ou serem anônimas
function soma(a, b) {
    return a + b;
} // variáveis fracamente tipadas (podem ser de qualquer tipo)
var elementoHTML = document.getElementById("teste");
```

Figura 9 – Algumas peculiaridades de Java Script

Fonte: O próprio Autor.

2.1.9 jQuery

No *Java Script*, precisamos muitas vezes acessar elementos do *HTML* e coletar valores, ou criar requisições assíncronas na página para determinar alguns resultados utilizando o servidor.

> *Java Script* tem um pequeno segredo embaraçoso: escrevê-lo é difícil... um programa feito no Chrome pode se tornar completamente errôneo no Internet Explorer 9.... É aí que entra o *JQuery*, uma biblioteca Java script que tem a intenção de deixar o Java Script mais fácil e divertido. (MCFARLNAD, 2011, p. 3).

Muitas dessas funcionalidades tem diferentes implementações em cada navegador.

Um dos problemas de Java script para requisições assíncronas é o uso do *XMLHttpRequest* ou *XHR*, observe esta requisição sendo feita na figura 10.

```
var client = new XMLHttpRequest();

client.open("GET", "unicorns-are-teh-awesome.txt", true);

client.send();

client.onreadystatechange = function() {

  if(this.readyState == 2) {

    alert("Deu certo");

  }

}
```

Figura 10 – Código para usar uma requisição assíncrona com *Java Script*

Fonte: (W3C, 2014, p. 1).

Agora veja como esta requisição pode ser mais facilmente implementada com o uso de *jQuery* na figura 11.

```
$.ajax({ url: "unicorns-are-teh-awesome.txt",

  success: function() { alert("Deu certo"); }

});
```

Figura 11 – A mesma funcionalidade com ajuda de jQuery

Fonte: Autor.

JQuery também conta com a ajuda do construtor de $, o qual utiliza um seletor do mesmo método do (Folha de estilos em cascata) CSS para selecionar elementos do corpo HTML. Expressões como *document.getElementById(“elemento”)* são simplesmente substituídas por *$(“#elemento”)*.

2.1.10 Folha de estilos

Para a W3C o CSS é um mecanismo simples de adicionar estilos (fontes, cores e espaçamento, por exemplo) ao documento WEB.

Em uma folha de estilos CSS (*Cascading Style Sheet*) temos dois aspectos: a seleção e aplicação de estilo. Pela W3C estes são os principais pontos que podemos controlar os elementos HTML:

a) Primeiro e último elemento.
b) Elementos pares ou ímpares.
c) Elementos específicos de um grupo.
d) Gradiente em textos e elementos.
e) Bordas arredondadas.
f) Sombras em texto e elementos.
g) Manipulação de opacidade.
h) Controle de rotação
i) Controle de perspectiva.
j) Animação.
k) Estruturação independente da posição no código HTML.

Utilizando os navegadores atuais temos a possibilidade de observar quais estilos estão sendo aplicados aos elementos do documento, com a ajuda das ferramentas de depuração, como se pode observar na figura 12.

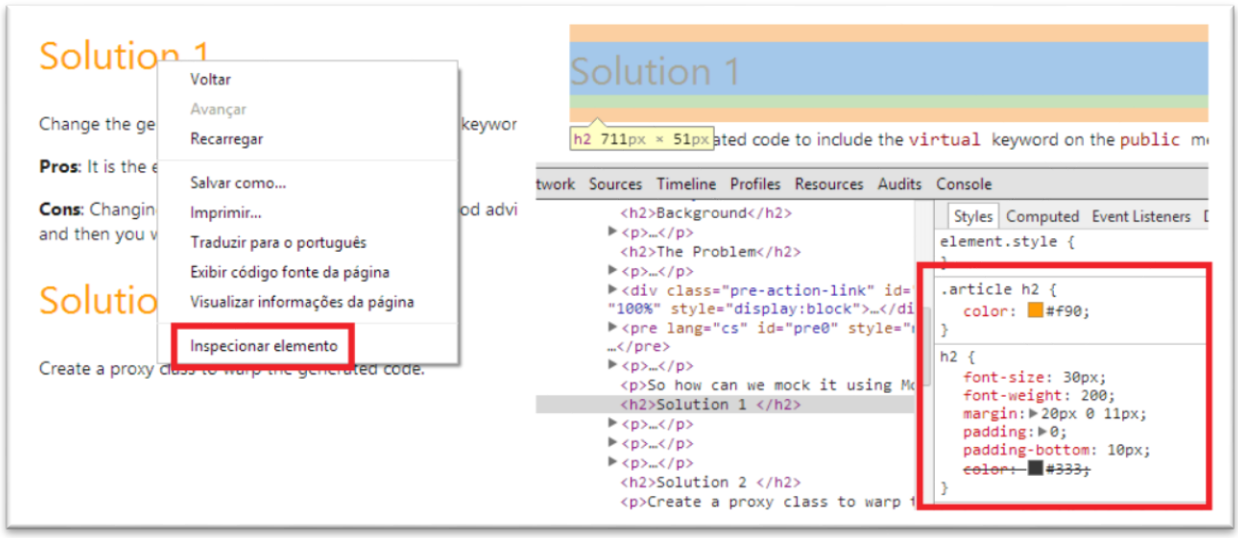

Figura 12 – Inspeção de elemento usando o navegador Chrome

Fonte: Autor.

A sintaxe do CSS é simples e auto explicativa:

```
seletor {
    propriedade: valor;
}
```

(W3C, 2014, p. 7)

Na figura 12, por exemplo, temos que o elemento *h2* terá:

a) Tamanho de fonte com 30 pixels.
b) Espessura da fonte em 200 (espessura relativamente normal)
c) Margem superior com 20 pixels de distância e inferior com 11 pixels.
d) Deslocamento inferior de 10 pixels, outros deslocamentos são nulos.

Também temos os elementos encadeados, nos quais a regra se aplica aos elementos "pai" ou elementos próximos como é o caso de:

```
.article h2 { }
```

Que se aplica somente aos elementos *h2* filhos de elementos da classe *article*. No caso se aplicaria à esta estrutura HTML:

```
<div class=”article”>
    <h2>Elemento Alvo do estilo</h2>
</div>
```

2.2 Design

2.2.1 Conteúdo de páginas web

Segundo Robert Martin (2011, p. 110), um dos itens mais importantes na criação de objetos, muito utilizado quando se programa em MVC é “da responsabilidade única”.

Segundo esse princípio, uma classe deve ser criada com o objetivo de resolver um problema isolado de todo o resto do sistema. Tal como um sistema de cadastro deve ter uma classe para gerenciar banco de dados, outra para interface do usuário e outra para controlar as ações do usuário e não uma que faça tudo em um único método cheio de laços e condições.

Partindo deste princípio temos a divisão do conteúdo Web, cada página do site será responsável por mostrar, através de uma *View*, um modelo de dados do sistema.

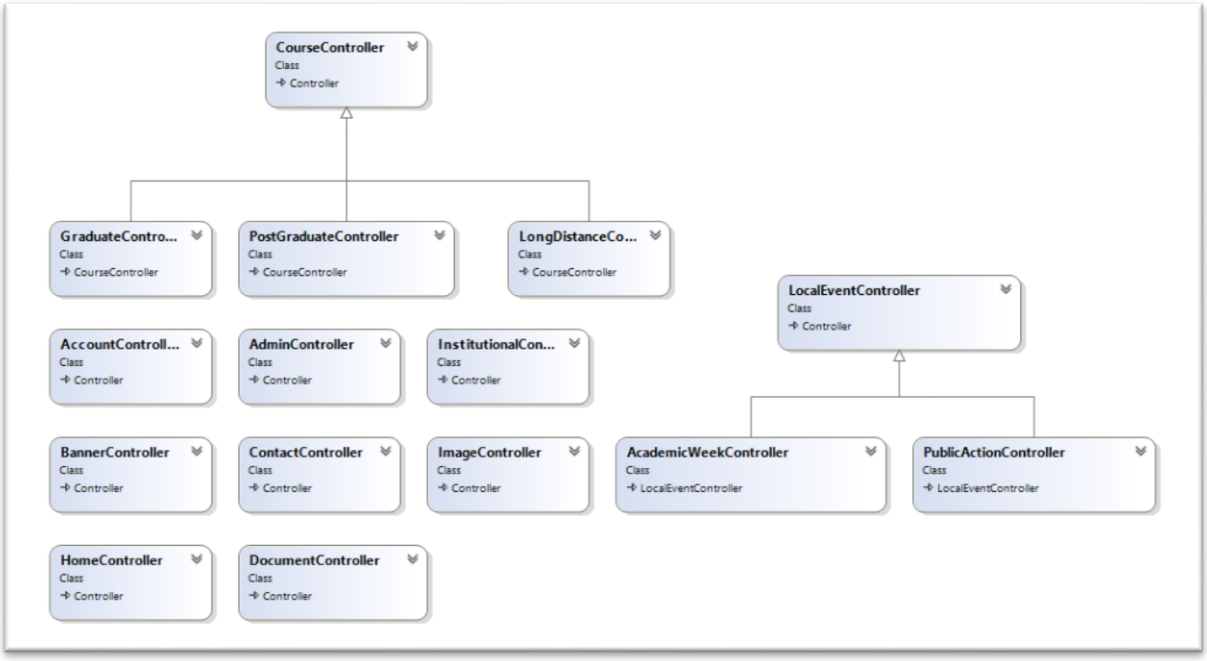

Figura 13 – Controladoras específicas para cada modelo do sistema

Fonte: O próprio Autor

Como se poder ver na figura 13, cada controlador se restringe unicamente ao seu modelo, as heranças aqui adicionadas têm como objetivo principal de reaproveitamento de código sendo que as mesmas possuem funcionalidades básicas aos modelos apresentados.

2.2.2 Padronização

Nós podemos assimilar informações de estilo a todos os elementos com a classe "*pastoral*" da seguinte forma:

```
*.pastoral { color: green }  /* elementos com class~=pastoral */
```

Ou simplesmente:

```
.pastoral { color: green }  /* todos os elementos com class~=pastoral */
```

O seguinte código assimila somente os elementos *H1* com *class*~="*pastoral*";

```
H1.pastoral { color: green }  /* Elementos H1 com class~=pastoral */
```

Dadas estas regras, a primeira instância de *H1* abaixo não terá o texto verde, enquanto o segundo terá.

```
<H1>Não verde</H1>

<H1 class="pastoral">Muito verde</H1>
```

A seguinte regra atribui qualquer elemento P, no qual o atributo *class* contendo valores separados por espaços, inclua "*pastoral*" e "*marine*" ao mesmo tempo.

```
p.pastoral.marine { color: green }
```

Essa regra se aplica quando o elemento possui *class="pastoral blue aqua marine"* mas não se aplica quando o elemento possuir *class="pastoral blue"*.

W3C (2014, p. 01)

A Partir dessa definição o projeto obedecerá a regra das classes CSS para padronização nos elementos do HTML, assim o usuário perceberá com maior facilidade o que é um título ou texto simples no documento, reconhecerá o os links também dentro do texto.

2.2.3 Cores

No presente projeto, não se pode pensar em criar algo que mude tudo e fuja das cores já registradas na sociedade, a instituição mantém a cor vermelha como principal artifício em seu logotipo e o branco como cor secundária.

É possível utilizar uma gradiente entre branco e preto em alguns casos, principalmente na parte administrativa do site, que não ficará exposta ao público.

2.2.4 Usabilidade

Segundo Andrade (2014, p1), a própria definição de usabilidade "é permeada por características subjetivas, que podem divergir entre usuários. Portanto, não existe uma receita única para se desenvolver um software perfeito no âmbito da usabilidade".

> Capacidade de um produto ser facilmente usado. (ABNT, 2002, p. 19)

Na área específica, web, temos que as páginas devem ser as mais simples possíveis, exigindo não mais que 2 ou 3 cliques para se chegar ao conteúdo desejado e não mostrar ao usuário o que não lhe interessa no momento.

O usuário deve ser capaz de, ao aberto o site, escolher rapidamente o conteúdo desejado, através de menus, banners ou links dentro do texto, como se pode observar na figura 14.

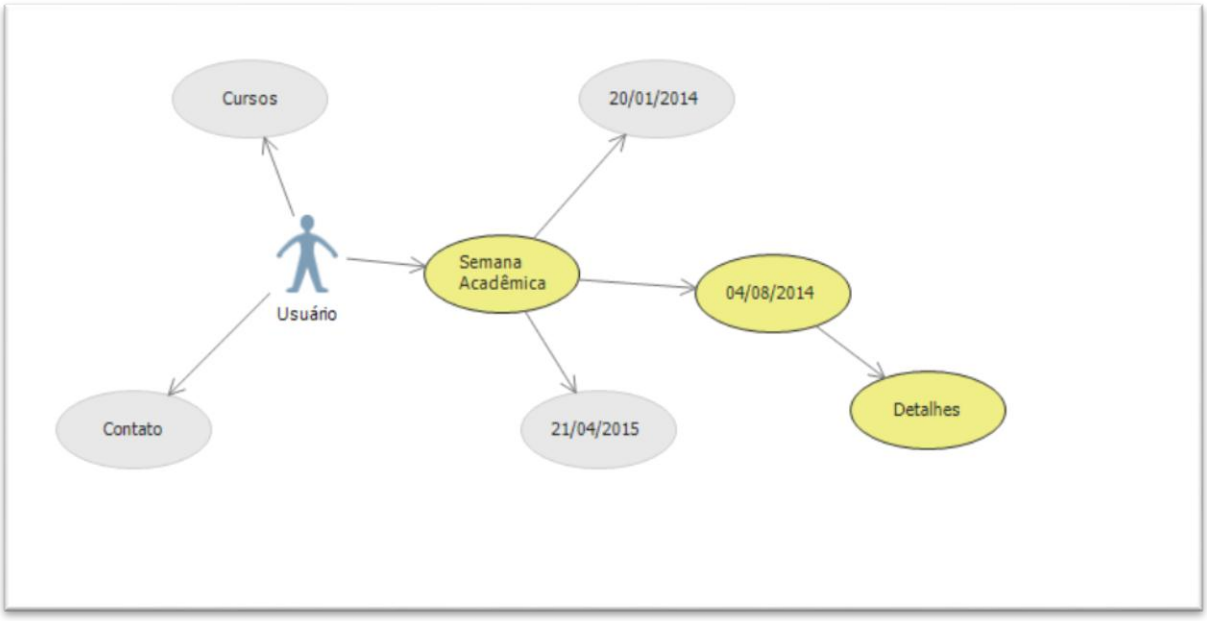

Figura 14 – Caso de uso exemplo na navegação entre links rápidos

Fonte: O próprio Autor.

2.2.5 Dispositivos móveis

Em uma universidade é necessário pensar que os usuários vão acessar o site de seus celulares e notebooks, muitas dessas telas não possuem alta resolução ou os navegadores podem ficar lentos devido a grande quantidade de elementos na página, ou elementos específicos como imagens. Pensando nisso será aplicado as principais páginas, regras CSS que melhorem a qualidade nos dispositivos móveis, removendo banners, imagens ou textos menos importantes.

2.3 Qualidades de serviço

2.3.1 Cache

Segundo a Microsoft, podemos utilizar um atributo em cada *Action* de uma Controller para melhorar a eficiência de saída de dados através do uso de Cache.

A saída de cache provê ao desenvolvedor um método muito fácil de aumentar dramaticamente a qualidade de uma aplicação ASP.NET MVC. Utilizando o atributo *OutputCache* para melhorar a qualidade das ações de *Controllers*. (Microsoft 2009, p. 1).

A documentação sobre o atributo está disponível em http://msdn.microsoft.com/en-us/library/system.web.mvc.outputcacheattribute(v=vs.118).aspx. Como, por exemplo:

```
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    [HandleError]
    public class HomeController : Controller
    {
        [OutputCache(Duration=10, VaryByParam="none")]
        public ActionResult Index()
        {
            return View();
        }

    }
}
```

Figura 15 – Exemplo de Cache

Fonte: Time de desenvolvedores da Microsoft (2009, p. 1).

A saída de dados da View Index, na figura 15, é guardada em cache por 10 segundos. No projeto o cache será utilizado em páginas estáticas e que demorem na

atualização pelo administrador do site. Serão utilizados tempos variados dependendo da situação, desde 10 segundos até 1 dia, por exemplo.

2.3.2 *Bundling* e minificação

No projeto temos, como de costume em um projeto *Web*, vários arquivos *CSS* e *Java Script* para customização do layout e acionamento de ações.

> *Bundling* e minificação são duas técnicas que você pode usar no ASP.NET 4.5 para melhorar o tempo de requisição dos recursos. Essas técnicas diminuem o tempo pela redução do número de requisições feitas ao servidor e reduzindo o tamanho do conjunto de recursos (como *CSS* e *Javascript*). (ANDERSON, R. 2012, p. 1).

	Usando B/M	**Sem B/M**	**Mudança**
Requisições	9	4	256%
KB enviados	3.26	11.92	266%
KB recebidos	388.51	530	36%
Tempo de download	510ms	780ms	53%

Quadro 3 – Diferenças na utilização de *Bundling* e Minificação.

Fonte: *(*Rick Anderson, 2012, p. 1).

No quadro 3 temos uma demonstração de otimização utilizando o *Bundling* e minificação, observe a diferença nos tempos e a quantidade de dados transmitida.

2.4 Usuário

2.4.1 Autorização

No presente projeto foi definido 4 (quatro) tipos de usuários, observe no quadro 4.

	Gerenciar Site	**Publicar**	**Mensagens (socialização)**	**Visualizar conteúdo**
Anônimo				x
Aluno			x	x
Professor		x	x	x
Administrador	X	x	x	x

Quadro 4 – Possíveis definições das funcionalidades dos usuários.

Fonte: O próprio Autor.

É claro que em um projeto onde teremos alunos, professores, administradores e usuários anônimos, temos que limitar as funcionalidades de modo a distinguir a função e necessidade de cada um no acesso ao sistema web.

2.5 Redes sociais

2.5.1 Troca de mensagens

Segundo RYAN (2011), uma rede social é uma simples aplicação hospedada em nossos computadores ou na rede, provendo um método de *login*, uma descrição do usuário e uma forma de se comunicar e interagir entre nossos amigos através da rede.

O que define a rede social é explícito a cada usuário que a utiliza, porém não precisamos seguir à risca tudo que uma rede social comum hoje, como *Facebook* e *Twitter* têm, é necessária apenas as troca de mensagens.

O professor e os alunos precisam interagir entre si, sobre assuntos da aula e compartilharem arquivos necessários às suas aulas anteriores ou futuras, faz parte de uma rede social os jogos, porém não convém a uma faculdade utilizar jogos em sua rede.

2.6 Eventos do servidor

Para que a troca de mensagens funcione perfeitamente precisamos de um método mais sofisticado de rede, já que o protocolo HTTP é voltado mais para uma única requisição, abrindo e fechando rapidamente uma conexão.

Como a rede está amplamente aberta na Internet, é necessário que o servidor administre as conexões e realize o roteamento das mensagens, assim cada cliente enviará e receberá as mensagens através do Proxy do servidor para com os seus usuários, veja na figura 16 o diagrama de caso de uso deste contexto:

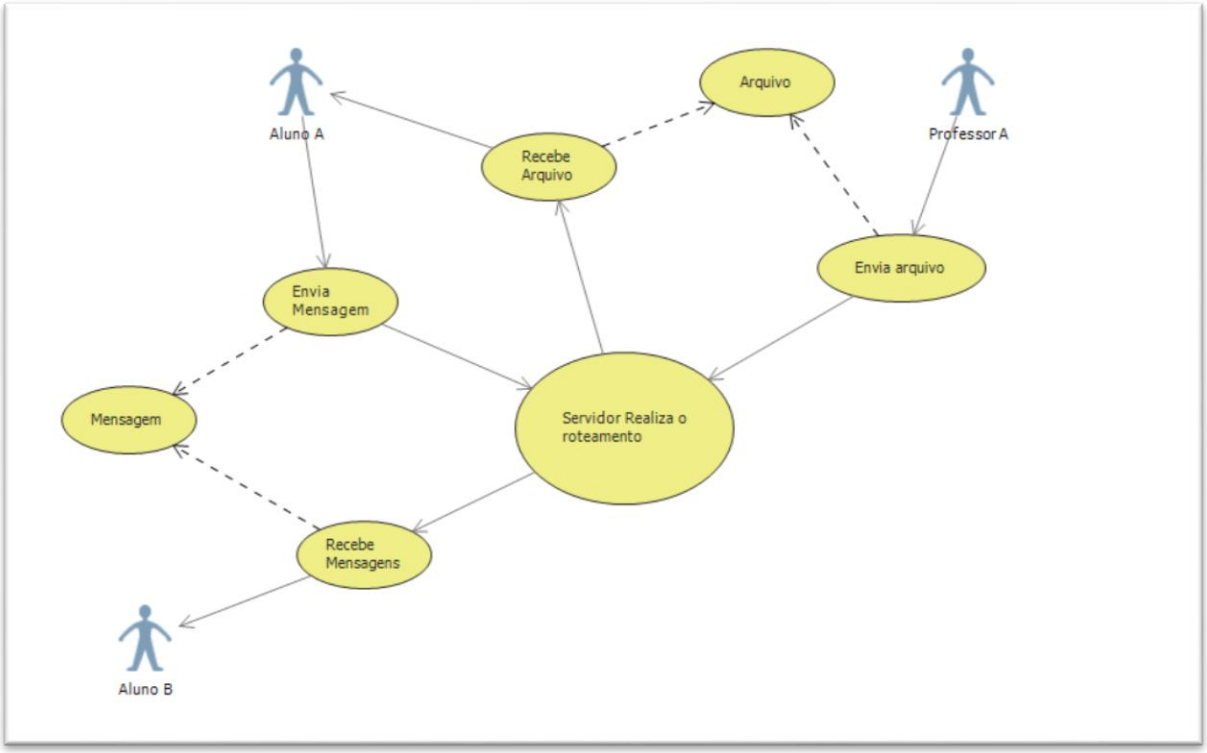

Figura 16 – Troca de mensagens

Fonte: O próprio Autor

O servidor nesse caso precisa ser capaz de enviar as mensagens ao cliente utilizando *Java Script*.

III. METODOLOGIA

O presente trabalho tem como base o conhecimento científico seguindo a pesquisa exploratória a fim de abordar o conhecimento utilizado para criar o sistema, inicialmente era necessário conhecer o sistema atual da instituição e verificar o que era necessário mudar e o que manter. Assim que constatado os fatos é iniciada uma pesquisa para obter métodos ágeis de desenvolvimento, busca de ferramentas e recursos. Depois de concluído a pesquisa se inicia o pré-projeto e a fase de desenvolvimento, tendo essa metade do tempo total do projeto. Será utilizado um cronograma de atividades para concluir a fase de pesquisa.

3.1 Cronograma

O cronograma de atividades segue como orientação do coordenador do curso Márcio Sembay como descrito no quadro 5.

Data	Assunto	OBS
18/03/2014	Desenvolvimento	Parte 1
25/03/2014	Desenvolvimento	Parte 2
01/04/2014	Desenvolvimento	Parte 3
08/04/2014	Introdução, Objetivos, Justificativa, Problemas.	Complemento
22/04/2014	Metodologia	Completa
29/04/2014	Resumo, Abstract, Resumen.	Completos
06/05/2014	Conclusão, Revisão de referências.	Completos
13/05/2014	Artigo	Parte 1
20/05/2014	Artigo	Parte 2
27/05/2014	Artigo	Parte 3
03/06/2014	Publicação	SBC
10/06/2014	Pré-apresentação	Em sala
17/06/2014	Pré-apresentação	Em sala
24/06/2014	Pré-apresentação	Em sala
01/07/2014	Ajustes finais	Em sala
07 à 11/07/2014	Defesa de bancas	Sala a ser definido

Quadro 5 – Cronograma TCC 1
Fonte: (SEMBAY, M., 2014).

3.2 Ferramentas

Para iniciar a fase de desenvolvimento serão utilizadas ferramentas para auxiliar a criação de módulos, manipular bancos de dados, testar, entre outras ações.

3.2.1 Visual Studio 2013

A ferramenta de ponta da Microsoft será utilizada como principal artifício durante todo o desenvolvimento e fase de testes do sistema Web, sua interface pode ser vista na figura 17.

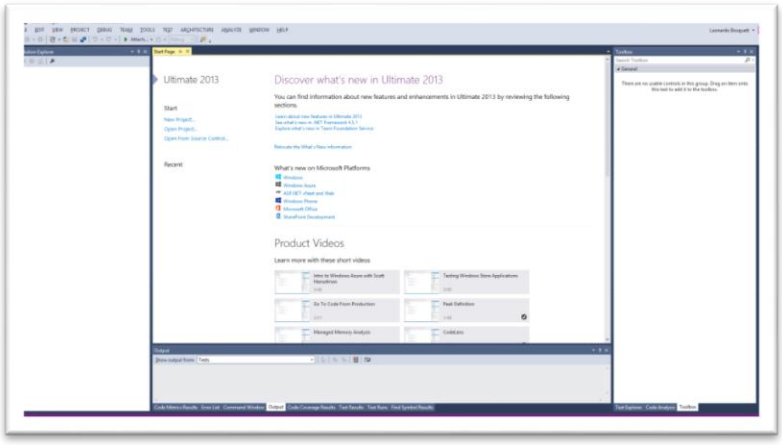

Figura 17 – Interface do usuário do Visual Studio 2013 Ultimate.

Fonte: O próprio Autor.

Possui ferramentas e um ambiente que facilitam o desenvolvimento de aplicações, *websites* e projeta o código para ser utilizado especialmente pelo *Internet Information Services* (IIS).

3.2.2 Internet Information Services (IIS)

Para hospedar a aplicação web, será utilizado o IIS, que atende a compatibilidade necessária entre as linguagens utilizadas, o ISS tem sua tela mostrada na figura 18.

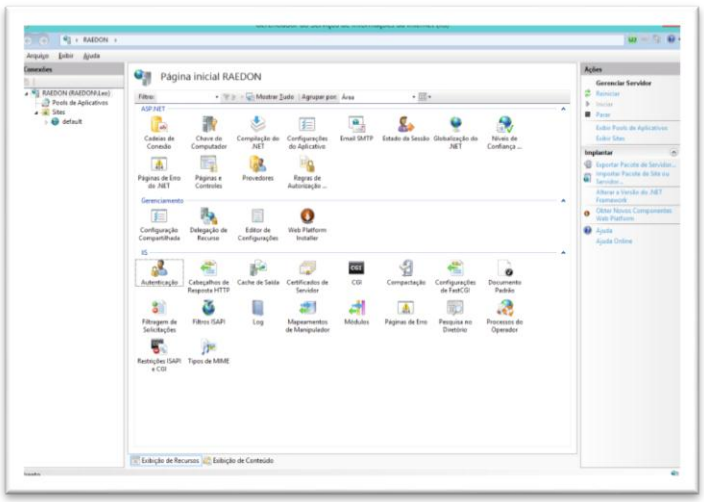

Figura 18 – IIS 8.0

Fonte: O próprio Autor.

3.2.3 SQL Server Management Studio

Todos os dados serão armazenados em banco de dados e arquivos de configuração específicos (no caso de configurações especiais e independentes), para gerenciamento e teste do banco de dados será utilizado o SQL Server Management Studio, sua interface pode ser vista na figura 19.

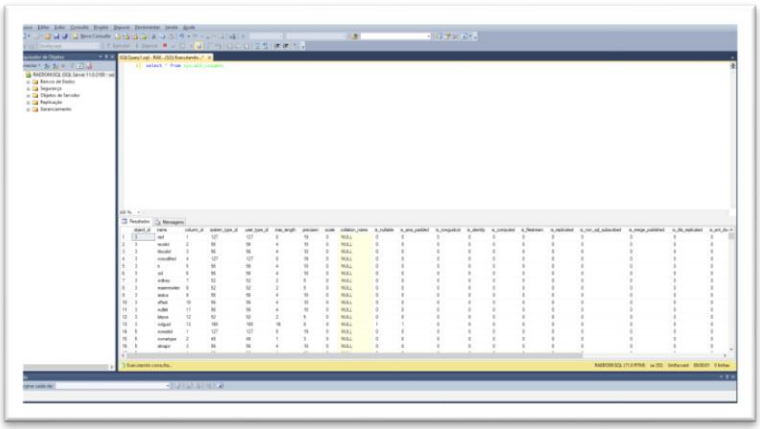

Figura 19 – SQL Management Studio 2012

Fonte: O próprio Autor.

3.2.4 Chrome

Para testes e visualização de páginas será utilizado o Chrome, a escolha foi feita baseada na qualidade de desenvolvimento do navegador na questão das tecnologias especificadas pela W3C e na grande quantidade de usuários que migram ao novo programa, o Chrome também habilita o modo de depuração, observe na figura 20.

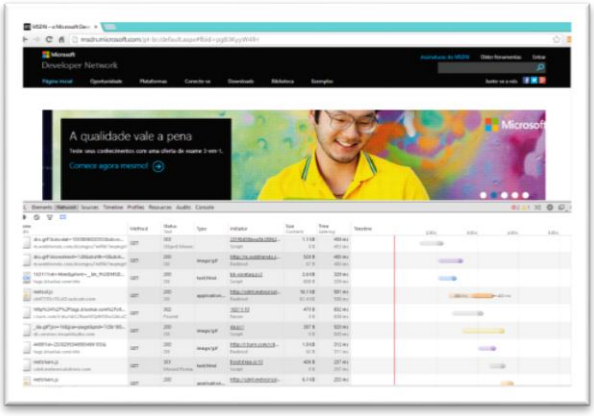

Figura 20 – Chrome, tempo de cada requisição de arquivo feita ao servidor

Fonte: HTTP://msdn.micrososft.com 06/03/2014

3.3 Ambiente

Todo o pré-projeto, fase de pesquisa e desenvolvimento foi ou será desenvolvido em uma máquina local com as seguintes especificações:

a) Plataforma x64
b) Processador Intel core i7 2600 3.4 ~ 3.7 GHz.
c) Memória ram DDR3 de 8GB.
d) Armazenamento de 1TB em disco rígido interno.
e) Processador gráfico ATI Raedon 6950.
f) Memória gráfica GDDR5 com 2 GB.

O sistema operacional instalado é Windows 8 Pro da Microsoft.

IV. PROJETO

4.1 Introdução

Com a utilização do MVC foi obtido uma melhoria de qualidade de código interno do site, bem como um melhor visual para atender os usuários.

4.2 Tela Inicial

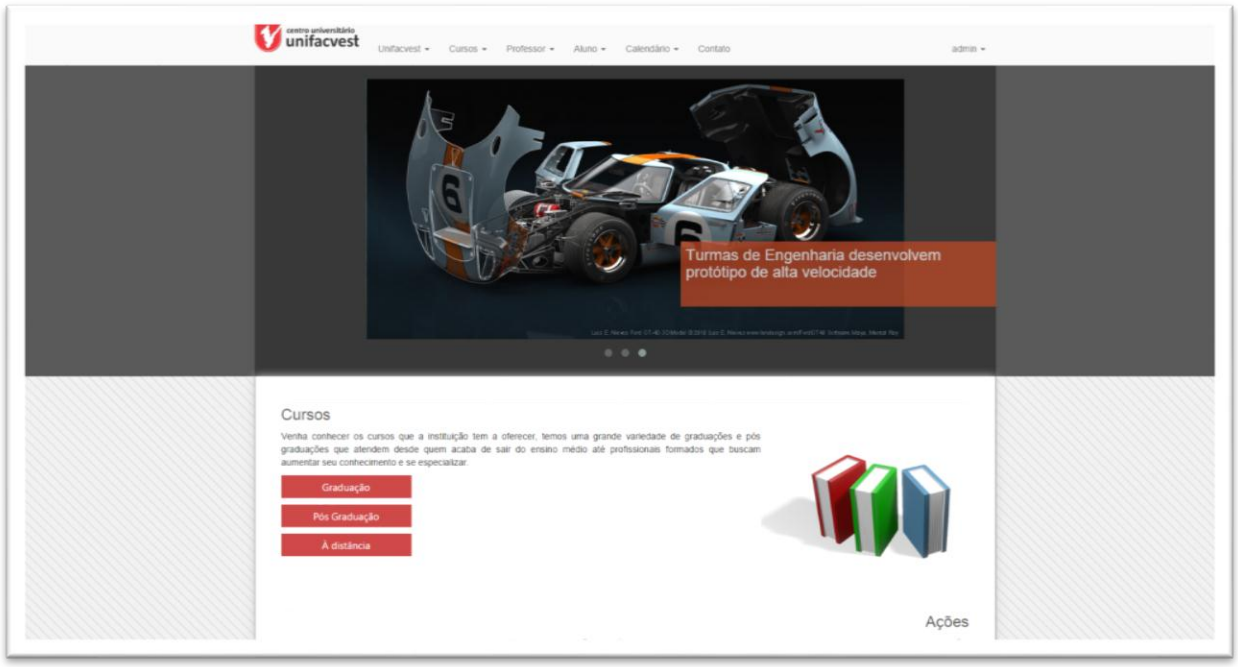

Figura 21 – Layout principal

Fonte: O próprio Autor

Os links foram distribuídos conforme descrito no quadro 6.

Menu	Sub menus
Unifacvest	Institucional, Documentos, Publicações, Páginas
Cursos	Graduação, Pós graduação, à distância
Professor	Sistema criador de provas, sistema de saúde
Aluno	Portal, Unimestre
Calendário	Eventos, Semanas acadêmicas
Contato	-

Quadro 6 – Links do menu superior

Fonte: O próprio Autor

Para concepção de um layout que pudesse ser reaproveitado por todas as outras telas do sistema, foi utilizado o próprio padrão do projeto ASP.NET MVC onde é possível identificar uma página inicial que serve de molde para todas às demais, esta página chama o método *RenderBody*() no qual inclui o corpo das outras e devolve a resposta HTML final, observe no quadro 7.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="shortcut icon" type="image/x-icon" href="/favicon.ico" />
    <title>@ViewBag.Title</title>
    <!-- inclusão de folha de estilos e escripts genéricos -->
    @Link.Reference(this, "~/Content/css")
    @Link.Reference(this, "~/bundles/admin/css")
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @Scripts.Render("~/Scripts/Admin.js")
</head>
<body>
    <div class="navbar navbar-static-top navbar-inverse">
        <div class="navbar-inner">
            <button class="navbar-toggle" data-toggle="collapse" data-target=".navbar-
collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav pull-right">
                    <li>
                        @Html.ActionLink("Sair", "LogOff", "Account")
                    </li>
                </ul>
                <ul class="nav navbar-nav">
                    <li>
                        @Html.ActionLink("Home", "Index", "Admin")
                    </li>
                    <li class="divider-vertical"></li>
```

```
                    <li>
                        @Html.ActionLink("Banners", "Index", "Banner")
                    </li>
                    <li class="divider-vertical"></li>
                    <li>
                        @Html.ActionLink("Institucional", "AdminIndex",
"Institutional")
                    </li>
                    <li class="divider-vertical"></li>
                    <li>
                        @Html.ActionLink("Documentos", "AdminIndex", "Document")
                    </li>
                    <li class="divider-vertical"></li>
                    <li>
                        @Html.ActionLink("Cursos", "AdminIndex", "Course")
                    </li>
                    <li class="divider-vertical"></li>
                    <li>
                        @Html.ActionLink("Eventos", "AdminIndex", "LocalEvent")
                    </li>
                    <li class="divider-vertical"></li>
                    <li>
                        @Html.ActionLink("Usuários", "Index", "User")
                    </li>
                    <li class="divider-vertical"></li>
                    <li>
                        @Html.ActionLink("Turmas", "Index", "ClassRoom")
                    </li>
                </ul>
            </div>
        </div>

    </div>
    <div class="admin-container">
        @if (IsSectionDefined("properties"))
        {
            <div class="properties page-section pull-left">
                <div class="inner">
                    @RenderSection("properties", false)
                </div>
            </div>
        }
        <div class="page-section page-section-large">
            <div class="inner">
                <!-- inclusão do corpo de outras páginas -->
                @RenderBody()
            </div>
        </div>
    </div>
    @RenderSection("scripts", required: false)
    @RenderSection("styles", required: false)
</body>
</html>
```

Quadro 7 – Layout do Administrador como exemplo de Layout em *CSHTML*

Fonte: O próprio Autor.

4.3 Painel Administrativo

A página do administrador do sistema possui um Layout mais simples, já que a finalidade desta página é apenas o gerenciamento.

Os menus ficaram agrupados na parte superior e cada um deles levará a área específica do modelo de dados, haverá uma área disponível à esquerda para gerenciamento de cada modelo, como visto na figura 22.

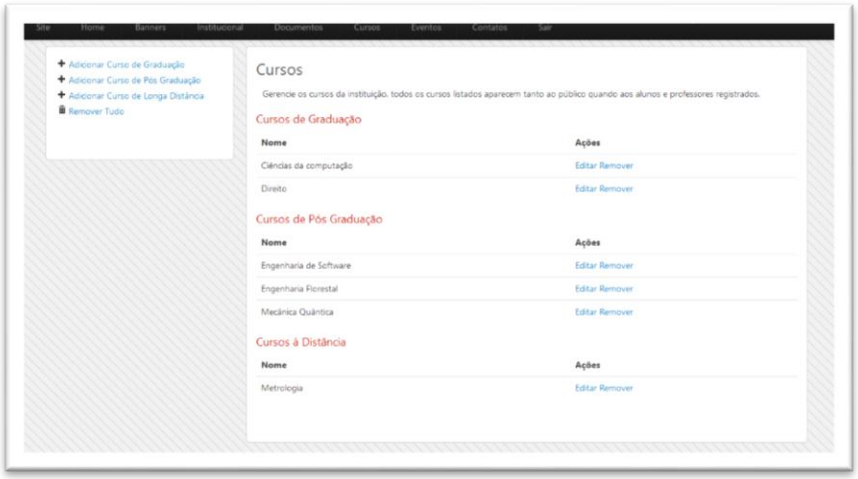

Figura 22 – Layout do administrador

Fonte: O próprio Autor

Com relação ao nível de segurança, os usuários possuem um nível de acesso, este nível de acesso é facilmente integrado com os filtros e anotações das controllers, sendo assim basta adicionar as anotações às classes que tem por objetivo realizar a gestão dos modelos do site que às mesmas vão obedecer aos filtros, conforme o quadro 8.

```
/// <summary>
/// Implementação dos filtros de segurança das Controllers
/// </summary>
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
public sealed class UnifacvestAuthorizeAttribute : AuthorizeAttribute
{
    #region Constantes

    public const string AdministratorRole = "Administrador";

    public const string AdministratorTeacherRoles = "Administrador, Professor";

    public const string AllRoles = "Administrador, Professor, Estudante";

    #endregion
```

```
        public UnifacvestAuthorizeAttribute(string roles)
        {
            if (string.IsNullOrWhiteSpace(roles))
            {
                throw new InvalidProgramException("Roles cannot be empty");
            }
            this.Roles = roles;
        }

        // -----------------

        protected override bool AuthorizeCore(HttpContextBase httpContext)
        {
            return this.AuthorizeUser(httpContext);
        }

        private bool AuthorizeUser(HttpContextBase httpContext)
        {
            if (httpContext != null && httpContext.User != null &&
                httpContext.User.Identity != null)
            {
                var name = httpContext.User.Identity.Name;
                using (var db = new UnifacvestDb())
                {
                    // seleção do usuário no banco através de seu logon
                    // registrado em sessão
                    IEnumerable<User> users = db.Users.Where(x =>
                        x.LogOnName.Equals(name));
                    if (users.Count() > 0)
                    {
                        // caso o usuário possua nível de acesso requerido
                        // e indicado pela controller
                        // o mesmo conseguirá realizar a ação
                        User user = users.First();
                        var roles = this.Roles.Split(',');
                        bool authorized = false;
                        foreach (var item in roles)
                        {
                            authorized = user.IsInRole(item.Trim());
                            if (authorized)
                            {
                                break;
                            }
                        }
                        return authorized;
                    }
                }
            }
            return false;
        }
    }
```

Quadro 8 – Filtro para controllers.

Fonte: O próprio Autor.

4.4 Portal

O layout do portal é diferente dos demais, já que a área necessária para publicações dos professores e listagem dos alunos da instituição é um pouco maior, como visto na figura 23.

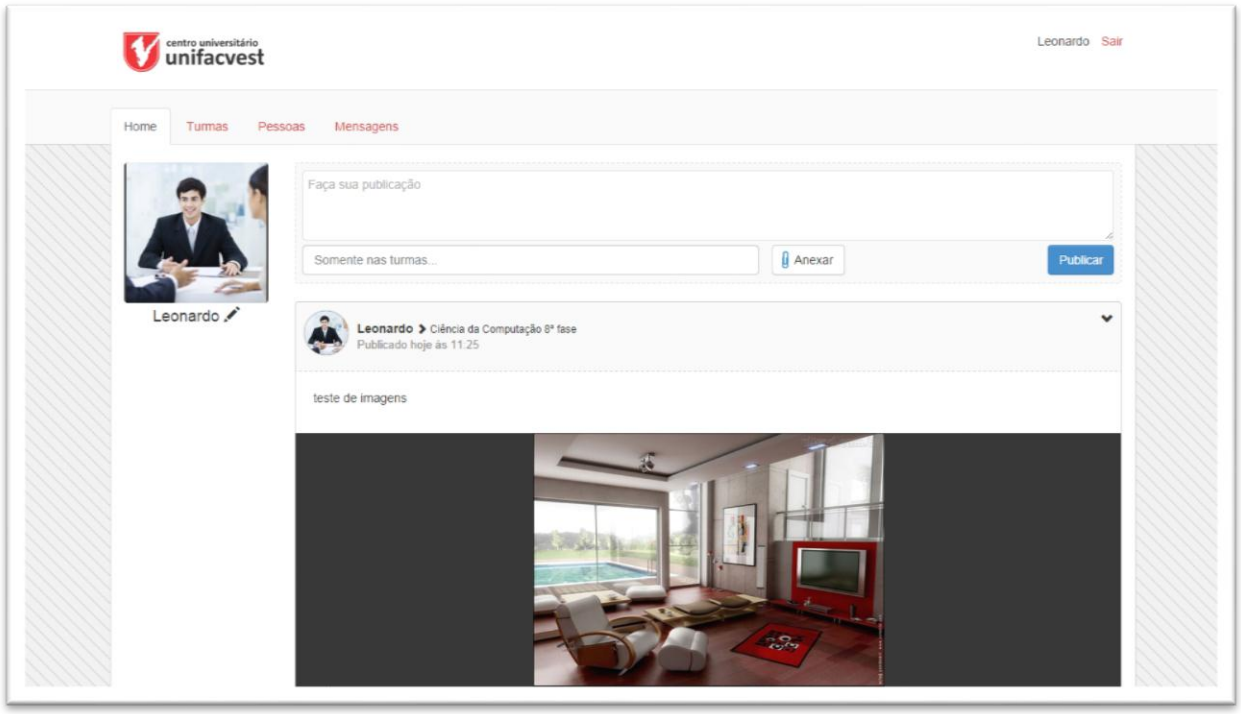

Figura 23 – Layout do portal

Fonte: O próprio Autor

Na página principal também são mostradas as principais publicações, como será visto no tópico adiante.

4.5 Publicações

O portal permite a publicação do usuário, o compartilhamento da mesma é visível à todos os usuários da instituição à menos que seja especificado uma turma, neste caso apenas os usuários que estão na turma poderão observar a publicação.

Os anexos quando possível possuem ícones de acordo com sua extensão, as extensões reconhecidas estão no quadro 9.

*.docx	Documento do MSO Word
*.pdf	Documento no formato portável
*.xlsx, *.xls	Planilha do MSO Excel
*.pptx, *.ppt	Apresentação de slides do MSO PowerPoint

Quadro 9 – Lista de extensões reconhecidas pelo Portal

Fonte: O próprio autor.

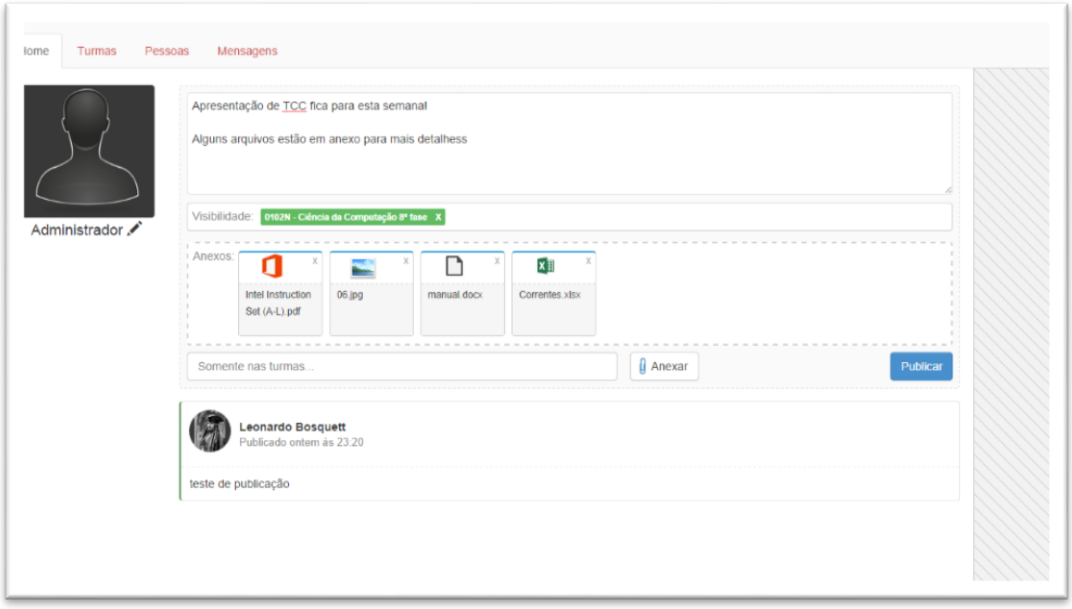

Figura 24 – Portal, quando o usuário está prestes a realizar uma publicação.
Fonte: O próprio autor.

Como visto na figura 24, uma publicação contém os seguintes itens:

- Texto
- Visão (totalmente público ou direcionado às turmas escolhidas)
- Anexos

As Publicações também são listadas na página inicial por ordem de data, são visíveis apenas as publicações realizadas em turmas em que o usuário está presente e as que têm visão pública (no caso de nenhuma escolha de turma ao publicar).

A figura 25 contém o diagrama de classes referente ao fluxo de publicação, é utilizado uma composição na *Controller* para utilizar corretamente a classe que trará as publicações mais recentes:

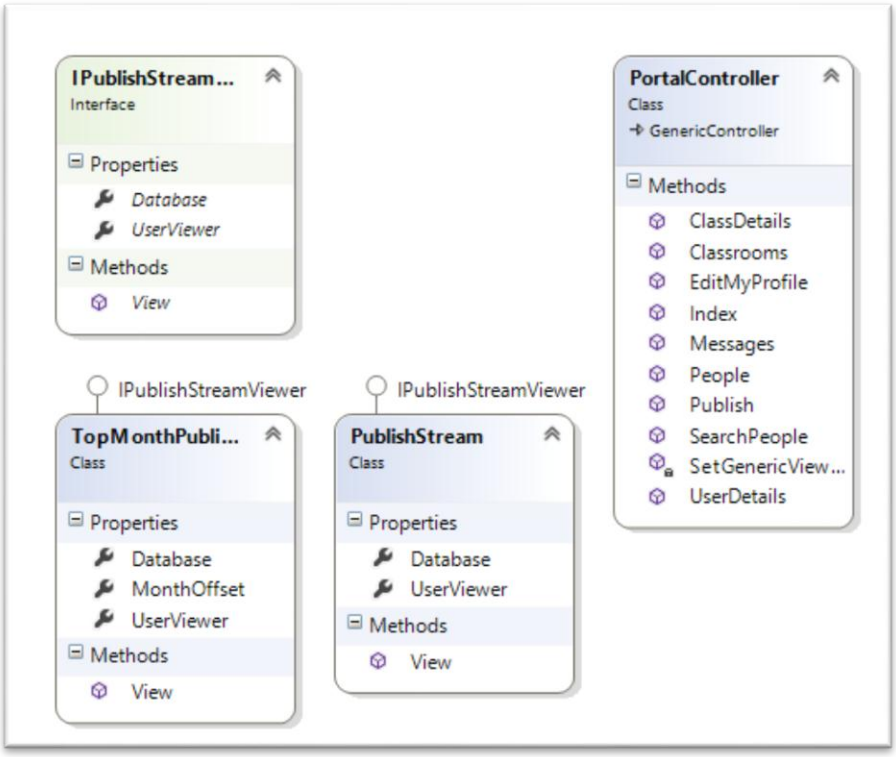

Figura 25 – Diagrama de classes do fluxo de publicação
Fonte: O próprio autor.

Na figura 26 é possível observar como este diagrama é trabalhado e relacionado com o código:

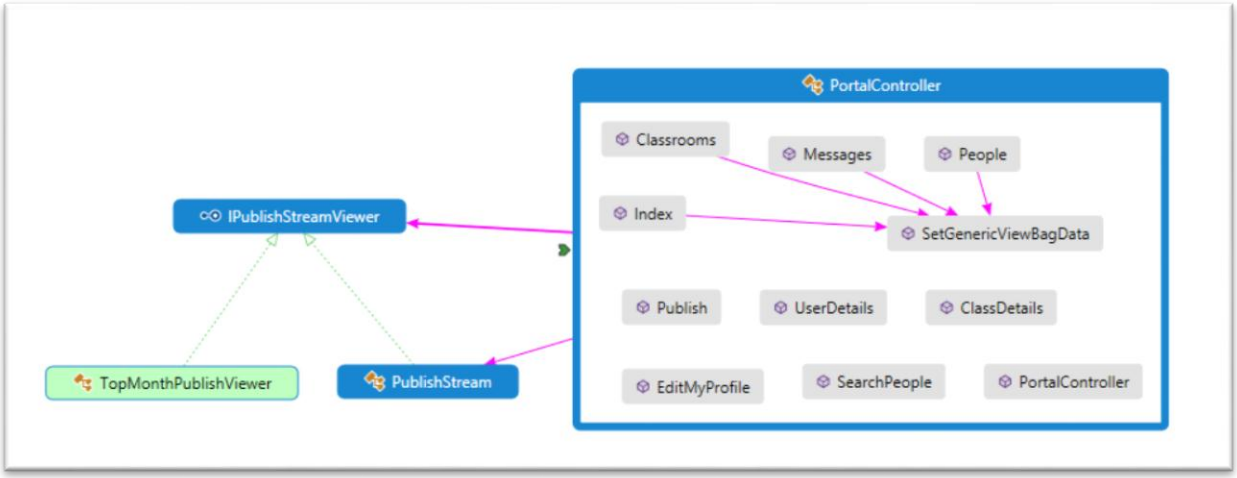

Figura 26 – Relacionamento entre *Controller* e a composição das publicações
Fonte: O próprio Autor.

E finalmente temos o código, no quadro 10, que realiza a busca das publicações, observe a simplicidade pela utilização do MVC em conjunto de padrões de projeto e auxílio de bibliotecas como Entity Framework que reduzem a complexidade de busca na camada de acesso à dados.

```
public class PublishUtilities
{
    public static IEnumerable<Publication> GetMonth(
        UnifacvestDb db, User userViwer, int monthBefore)
    {
        IList<Publication> ret = new List<Publication>();

        DateTime startCut = DateTime.Now.AddMonths(monthBefore - 1);
        DateTime endCut = DateTime.Now.AddMonths(monthBefore);

        var publications = db.Publications
                .Include(x => x.ClassRooms.Select(r => r.Course))
                .Include(x => x.Author)
                .Where(x =>
                    x.Date > startCut &&
                    x.Date <= endCut)
                .OrderByDescending(x => x.Date)
                .ToList();

        IList<int> userClassRoomIds = (from ClassRoom classRoom in
                                       userViwer.ClassRooms
                                       select classRoom.Id).ToList();

        foreach (var item in publications)
        {
            // Visão pública
            if (item.IsPublic())
            {
                ret.Add(item);
            }
            else
            {
                // Relacionado às turmas do usuário
                foreach (ClassRoom classRoomItem in item.ClassRooms)
                {
                    if (userClassRoomIds.Contains(classRoomItem.Id))
                    {
                        ret.Add(item);
                    }
                }
            }
        }

        return ret;
    }
}
```

Quadro 10 – Código relacionado ao fluxo de publicações

Fonte: O próprio autor.

4.6 Usuários

Uma das grandes utilidades deste projeto é a rápida procura por pessoas tanto na página dedicada ao serviço quanto nas páginas das turmas, o motivo: fácil descoberta do contato nas fases iniciais ou mesmo o contato de emergência entre aluno e professor e vice-versa, observe na figura 27.

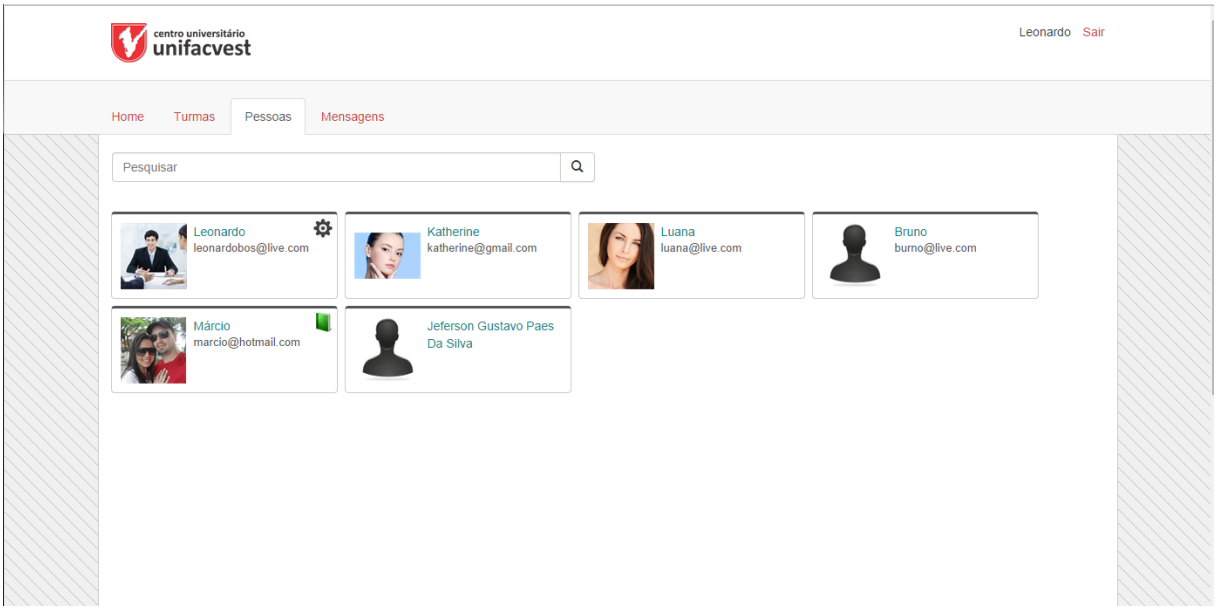

Figura 27: Página de usuários

Fonte: O próprio autor.

O método de busca é feito via *AJAX* com a pesquisa dos campos como nome e email através de expressão regular, utilizando o código no quadro 11:

```
function doSearch() {
    $.ajax({
        url: "/Portal/SearchPeople",
        dataType: "json",
        type: "post",
        data: {
            query: $("#search-names").val()
        }
    }).done(function (response) {
        var t = $(".people-data").html("");
        for (var x = 0; x < response.length; x++) {
            var item = response[x];
            var bg = "background-image: url('/User/Thumbs?un="+ item.logonName + "')";
            var role_content = "";
            if (item.role == "admin") {
                role_content = "<div class='role admin-role'></div>";
            }
            else if (item.role == "teacher") {
                role_content = "<div class='role teacher-role'></div>";
            }
            t.append('<a class="people-item" href="/@@' +
                item.logonName + '" data-id="' + item.id + '">\
                <div class="photo" style="' + bg + '"></div>\
```

```
                <div class="contact-info">\
                    ' + role_content + '\
                    <div class="name">' + item.name + '</div>\
                    <div class="email">' + item.email + '</div>\
                </div>\
            </a>');
        }
    });
}
```

Quadro 11: Pesquisa com AJAX dos usuários
Fonte: O próprio autor

4.7 Turmas

Existe uma página no portal dedicada à visualização das turmas (estas podem ser gerenciadas na página de administração somente). Cada turma possui seu código (opcional), não há pesquisa nesta página, embora tenha um agrupamento por curso, o que facilitará a procura dos alunos. O título dos cursos tem uma cor específica para seu tipo, obsrrve no quadro 12:

Tipo	Cor
Graduação	Normal (Preto)
Pós-Graduação	Azul
À distância	Ciano

Quadro 12: Cores para os títulos dos cursos
Fonte: O próprio autor

Observe na figura 28 um exemplo desta página com alguns cadastros feitos:

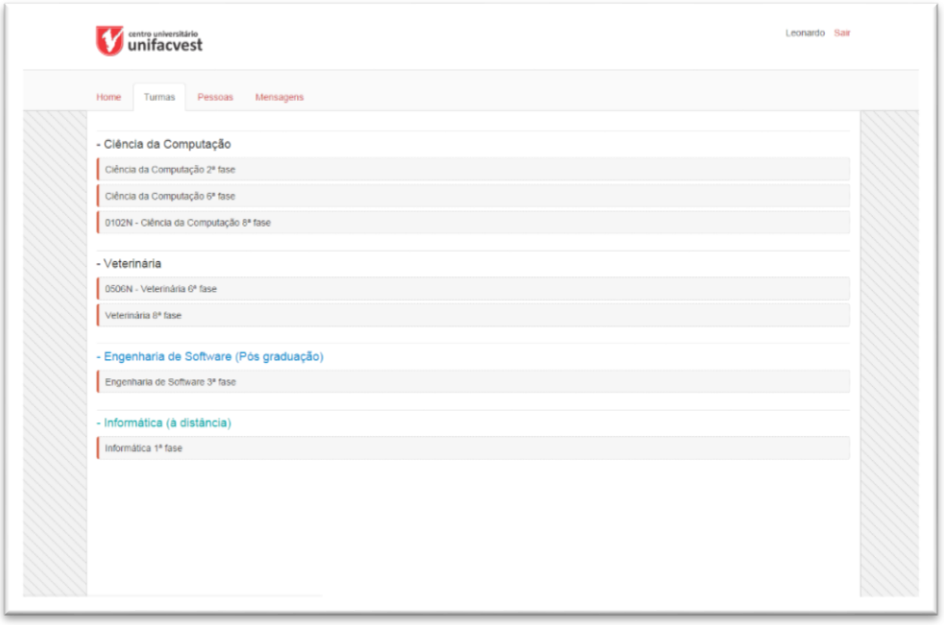

Figura 28: Página de cursos com alguns cadastros
Fonte: O próprio autor.

A fase é gerada dinamicamente conforme a data de início especificada na página de administração.

4.8 Integração do sistema de saúde (projeto específico)

Devido à necessidade específica da instituição, foi elaborado um projeto para gestão de pacientes do sistema de saúde. O projeto possui suas próprias regras de negócio o que não implicou de forma alguma na integração com o sistema devido à separação de código trabalhada do MVC.

Na figura 29 é possível observar o modelo de código por referência criado.

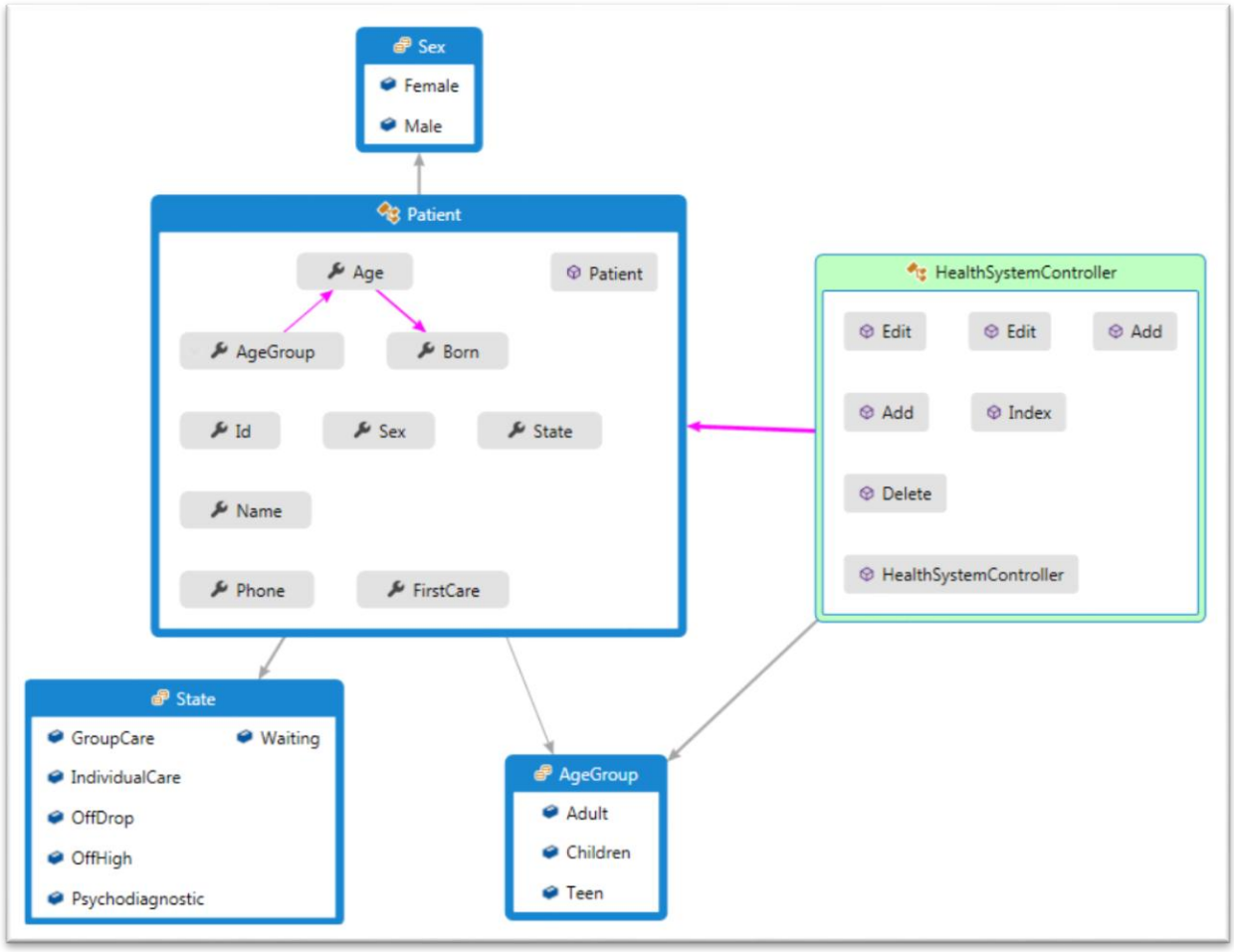

Figura 29: Diagrama de código por referência do Sistema de saúde
Fonte: O próprio autor.

Na figura 30 é detalhada a tela de gestão dos pacientes, a pesquisa inclui todos os campos, facilitando a experiência do professor.

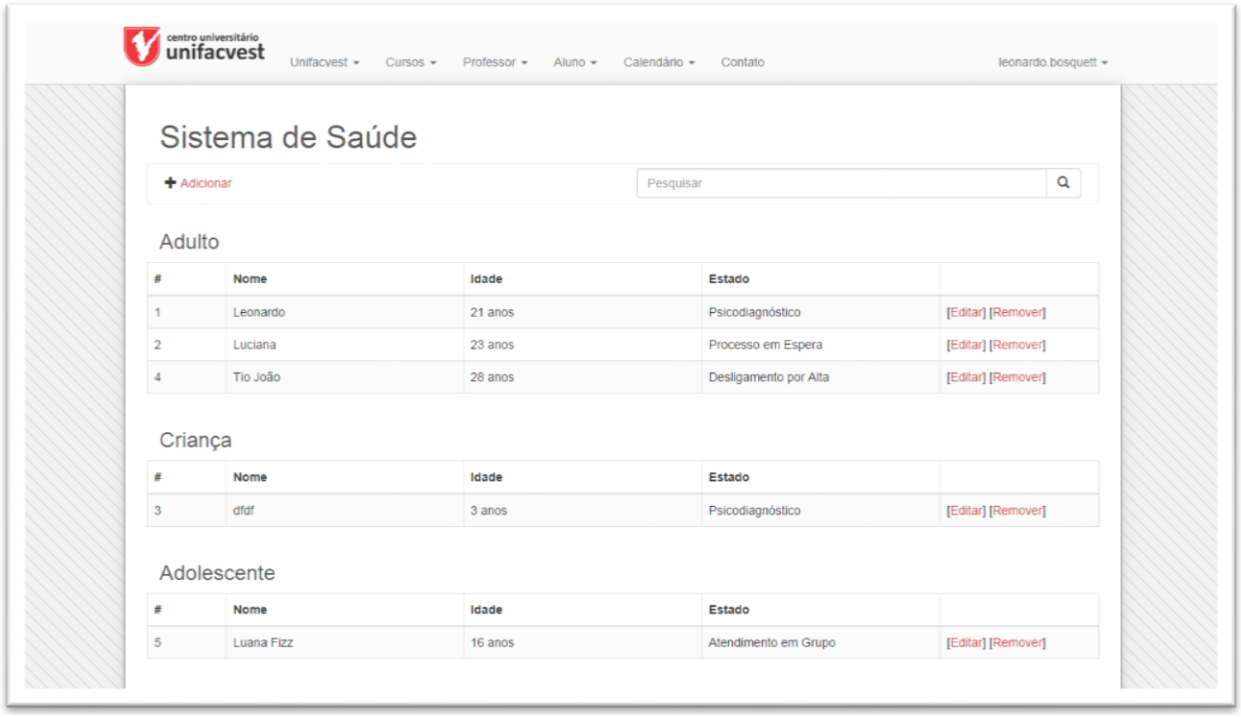

Figura 30: Gestão dos pacientes
Fonte: O próprio autor.

A gestão deste sistema fica responsável somente pelos professores e administradores, nenhum aluno consegue acessar esta página nem mesmo realizar alteração dos dados.

A busca é feita por *javascript* para obter performance de resultado. Observe no quadro 13 o código de busca:

```
function genericFind() {
    var patients = $(".patient-search-data tr");

    var reg = new RegExp($("#search-names").val().toLowerCase());

    for (var x = 0; x < patients.length; x++) {
        var i = $(patients[x]).addClass("hidden");
        if (reg.test(i.attr("data-name").toLowerCase()) ||
            reg.test(i.attr("data-age").toLowerCase()) ||
            reg.test(i.attr("data-state").toLowerCase())) {
            i.removeClass("hidden");
        }
    }
}
```

Quadro 13: Código de busca por expressão regular do sistema de saúde
Fonte: O próprio autor.

4.9 Software de auxílio ao professor

Um trabalho de conclusão de curso paralelo a este, a integração não foi realizada, devido ao desenvolvimento ser separado. Foi criada uma página visível apenas aos professores que disponibiliza o software para download. O modelo de apresentação foi criado conforme a figura 31:

Figura 31: Página de download do Sistema criador de provas
Fonte: O próprio autor.

Ao clicar em download, será baixado um arquivo contendo os componentes da aplicação necessários para utilização do sistema, nessa mesma página é possível ver os requisitos do software e funcionalidades disponibilizadas por Carlos G. Coelho.

4.10 Mensagens

O Sistema conta com um mensageiro instantâneo, utilizando SignalR é possível enviar e receber mensagens de outros alunos, professores e administradores que estão inclusos nas mesmas turmas em que o usuário está, veja na figura 32.

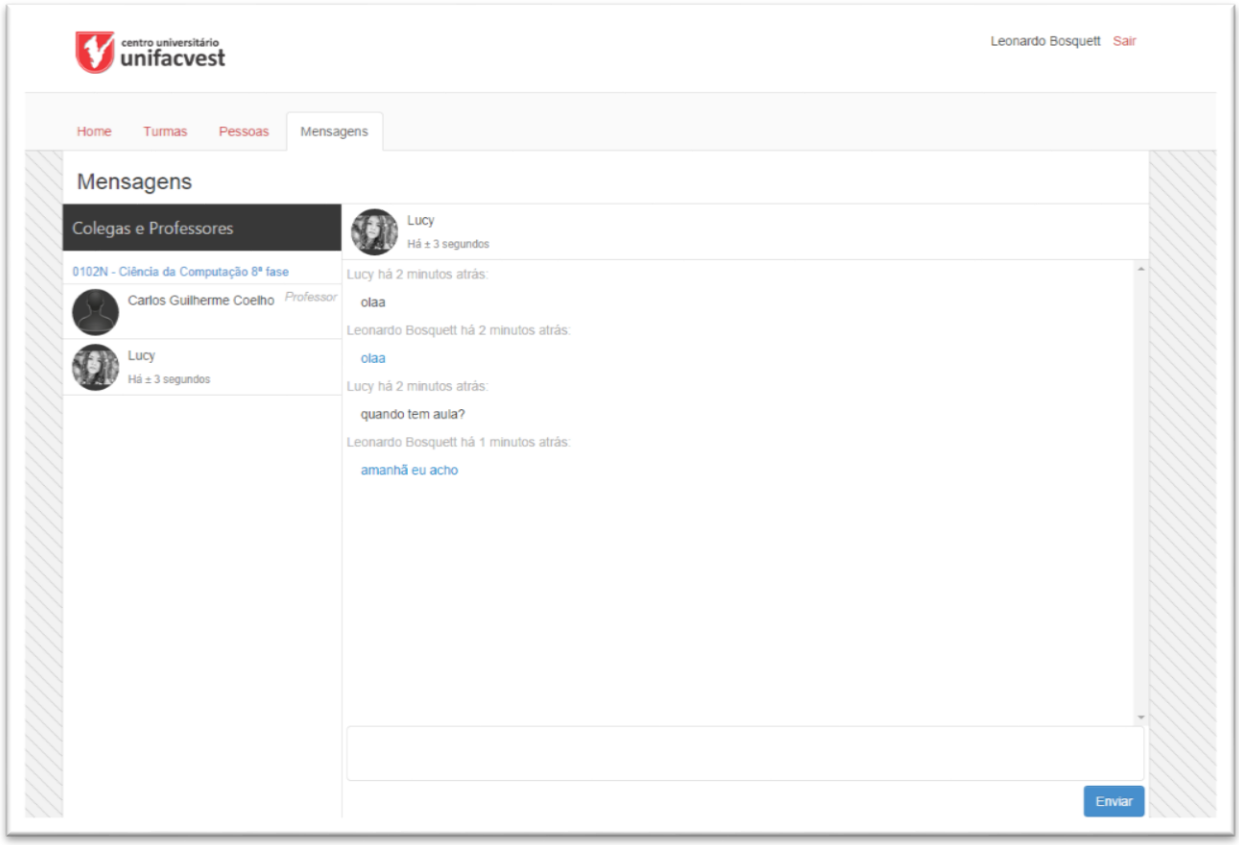

Figura 32: Mensageiro instantâneo
Fonte: O Próprio autor.

As mensagens são armazenadas de acordo com o modelo de classe no quadro 14:

```
public class Message
{
    [Key]
    public int Id { get; set; }

    [DataType(DataType.DateTime)]
    [Index(IsClustered = false)]
    public DateTime Sent { get; set; }

    [Index(IsClustered = false)]
    public int FromId { get; set; }

    [ForeignKey("FromId")]
    public User From { get; set; }

    [Index(IsClustered = false)]
    public int ToId { get; set; }
```

```
    [ForeignKey("ToId")]
    public User To { get; set; }

    [MaxLength(400)]
    [Required]
    public string Text { get; set; }

    public bool Viewed { get; set; }

    public static string FormatSentDate(DateTime sent)
    {
        DateTime now = DateTime.Now;
        TimeSpan diff = now.Subtract(sent);
        if (diff.TotalSeconds < 60)
        {
            return "há menos de um minuto";
        }
        else if (diff.TotalMinutes < 60)
        {
            return string.Format("há {0} minutos atrás", (int)diff.TotalMinutes);
        }
        else if (diff.TotalHours < 60)
        {
            return string.Format("às {0:HH:mm}", sent);
        }
        else return
            string.Format("em {0:dd/MM/yy} às {0:HH:mm}", sent);
    }

    public string ToSentString()
    {
        return Message.FormatSentDate(this.Sent);
    }
}
```

Quadro 14: Classe das mensagens
Fonte: O próprio autor.

Observe também que há um método *FormatSentDate*, este serve para indicar o quanto tempo a mensagem foi enviada ao usuário, esse tempo é aproximado (devido fuso horário, tempo da requisição e resposta em ambos os clientes) e calculado no lado do servidor.

4.11 Diagramas do Sistema

Foram definidos três diagramas básicos, o primeiro se refere ao modelo de dados e inclui os elementos associados ao banco e é o quesito inicial para separação das telas de apresentação do site incluindo a administração do sistema, isto pode ser visualizado na figura 33.

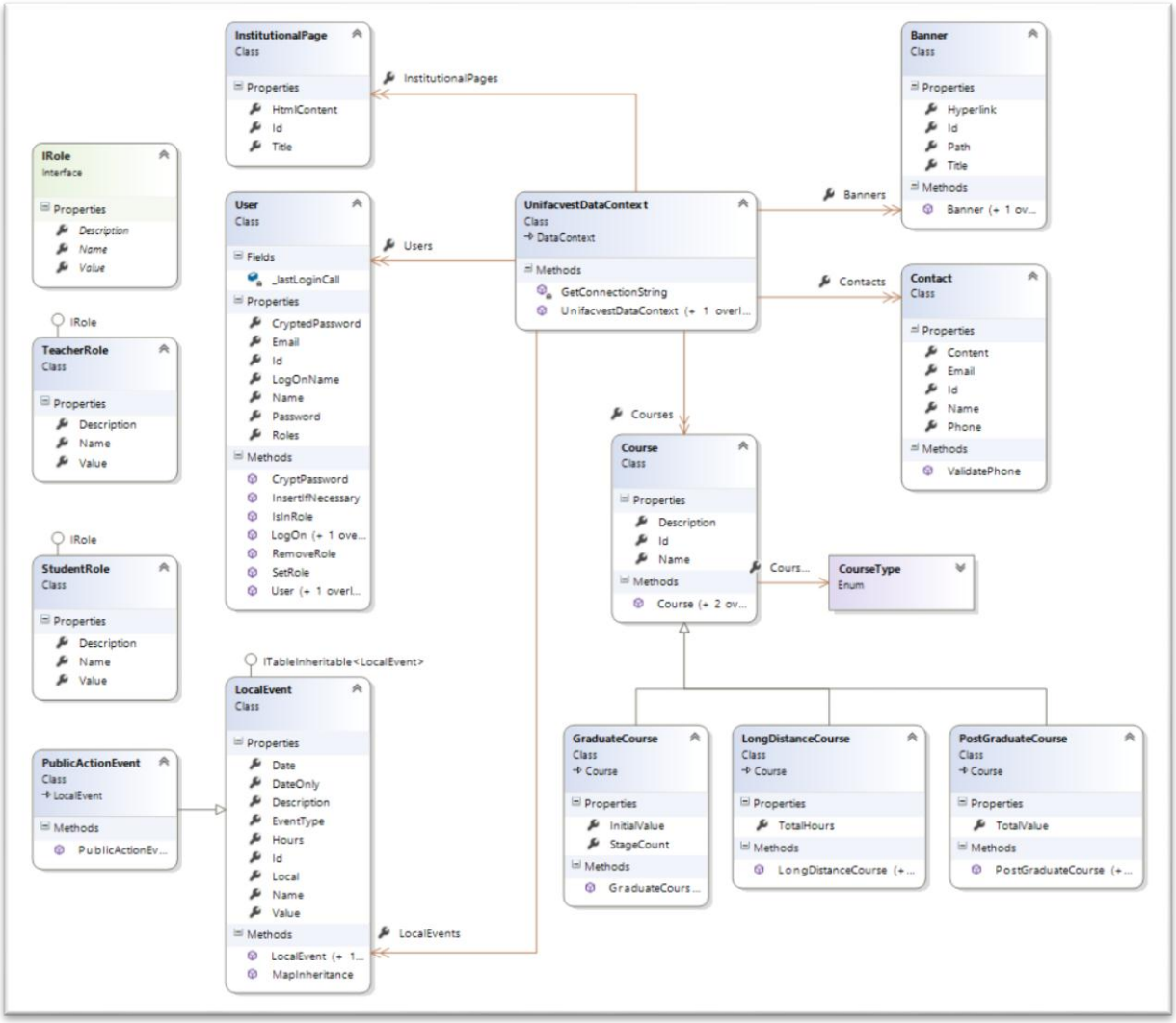

Figura 33 – Diagrama de classes gerais do sistema
Fonte: O próprio autor

Este é o diagrama de caso de uso do sistema, descrito na figura 34:

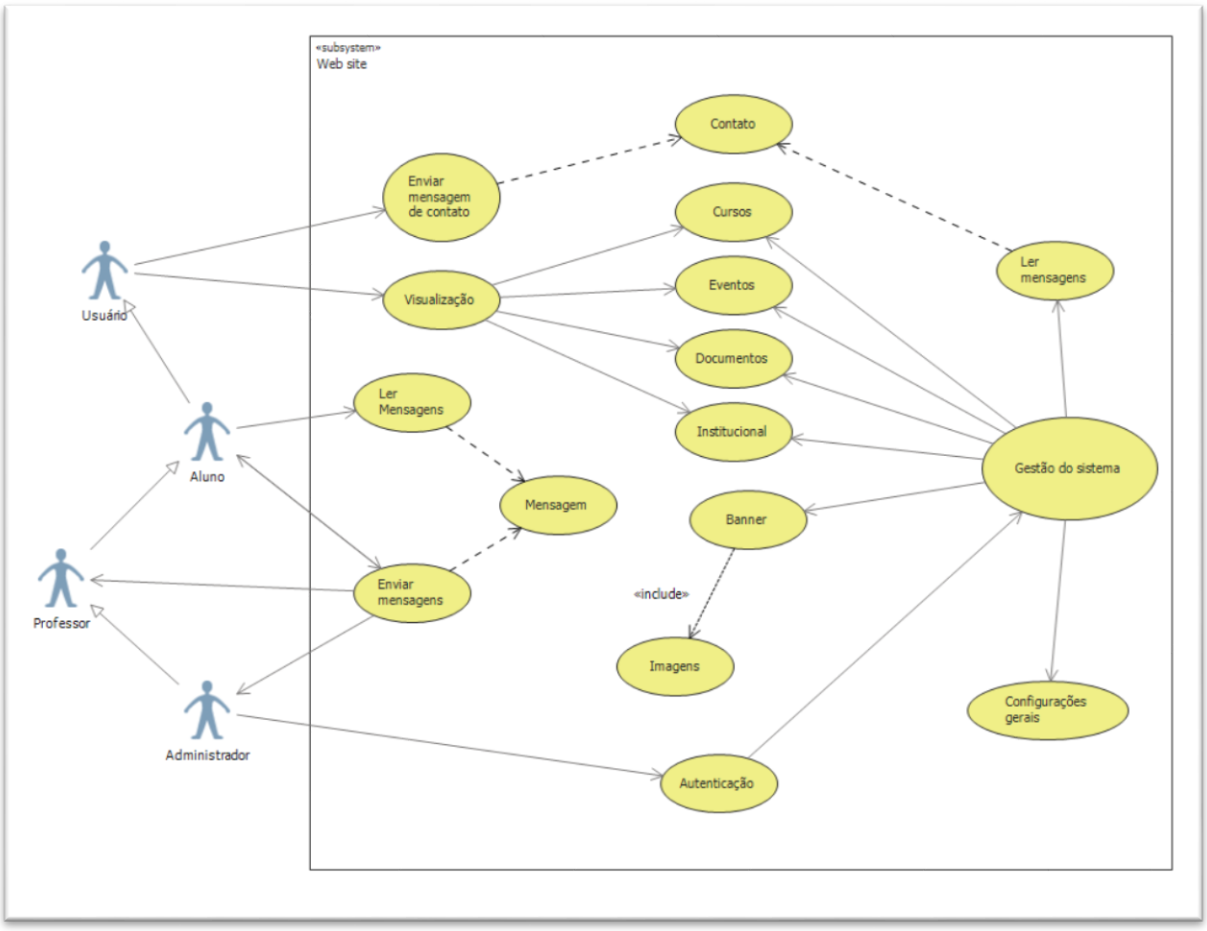

Figura 34 – Diagrama de caso de uso geral do sistema

Fonte: O próprio autor

Cada usuário tem seu nível de acesso dentro do sistema (um é a generalização de outro). O usuário tem o nível mais baixo de acesso enquanto que o administrador possui todos os privilégios disponíveis, seguido de professor a aluno.

A principal diferença entre professor e aluno é o acesso à aplicativos externos como a aplicação desenvolvida por Carlos Guilherme Coelho.

O diagrama de atividades é simplificado devido o uso do MVC, assim todas as requisições de todos os usuários do sistema são barradas pelo padrão de projeto, não é necessária uma segunda intervenção como validação do fluxo de dados ou tratamento de erros intensivos, como visto na figura 35.

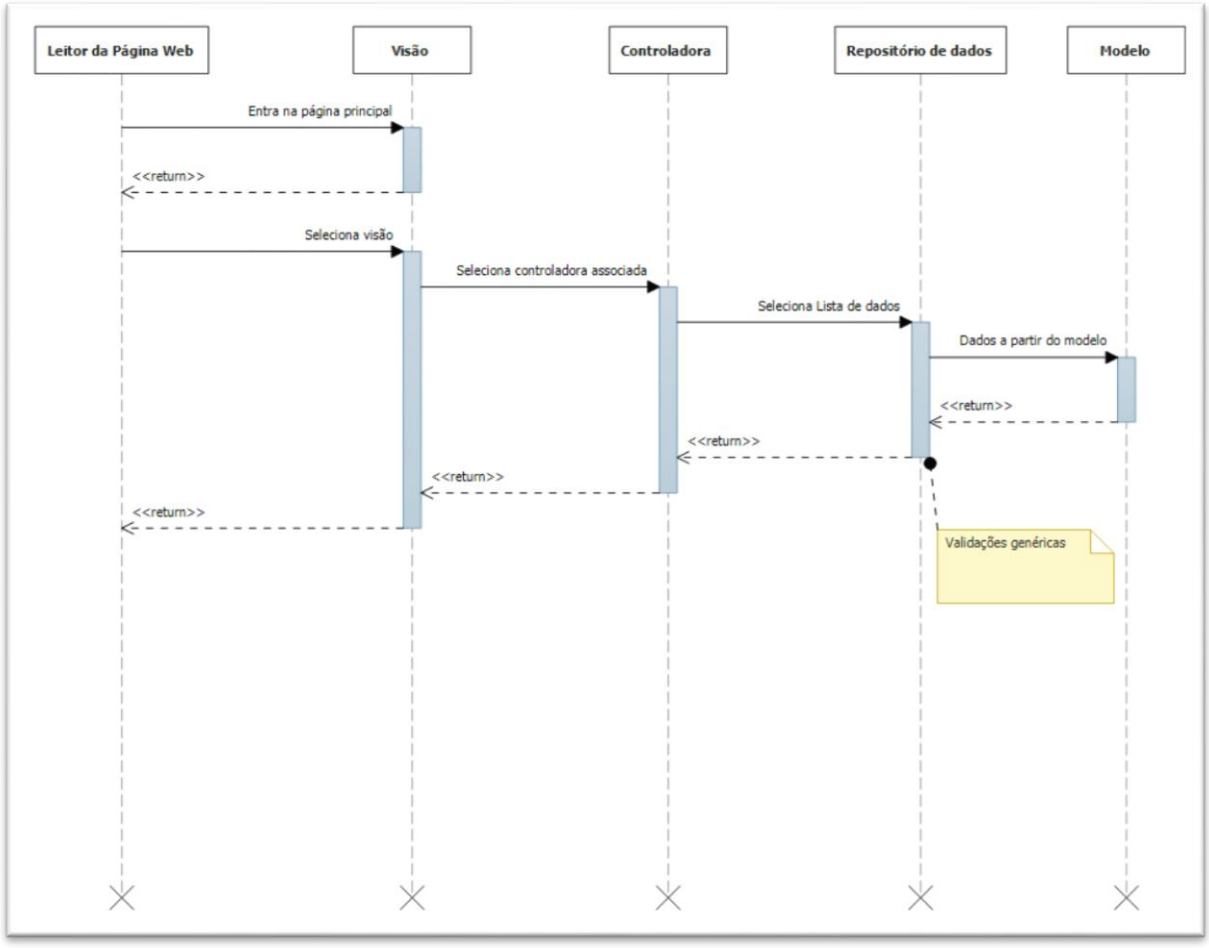

Figura 35 – Fluxo de atividade do sistema

Fonte: O Próprio Autor

5 CONCLUSÃO

Com a utilização dos padrões de projeto aqui descritos, MVC e *Observer*, por exemplo, foi possível chegar ao objetivo proposto já que a utilização destes aumenta em muito a qualidade do sistema, o design está sujeito a alterações num futuro próximo, pois este é um quesito volátil de qualquer sistema ou projeto com interface de usuário. Quanto à utilização observa-se que o site é informativo e administrável internamente, estendendo sua funcionalidade quando se fala na rede social entre os integrantes da instituição.

6 TRABALHOS FUTUROS

O armazenamento das mensagens hoje está em um banco relacional Sql Server. Não é preciso testes para saber que uma alta quantidade de linhas é inserida na tabela a cada conversa, para uma pequena amostra (abaixo de 1 milhão de linhas nesta tabela) o impacto no desempenho do servidor é quase nulo, mas para uma grande utilização deste serviço (algo em torno de 15~100 milhões de mensagens) o servidor pode ficar lento devido à carga de memória. Uma rápida solução para isto seria escalar o ambiente, aumentando a quantidade de servidores de banco de dados, esta questão pode ser também resolvida aplicando um armazenamento NoSQL, utilizando o Azure ou o MongoDB.

Futuramente será necessário recriar o design do site, para isto a única camada que precisa ser refeita é a View, a lógica do site é reaproveitável devido a utilização do MVC.

O mensageiro pode ser aprimorado para enviar mensagens em grupos.

7 REFERÊNCIAS

ROCHELEAU, Jake. **Discussão sobre o site da instituição Unifacvest** 03/03/2014. Entrevista concedida à Leonardo (autor).

BALTZAN, Paige; PHILLIPS, Amy. **Sistemas de Informação**. Brasil: McGraw, 2012. 384 p.

FREEMAN, Adam**. Pro ASP.NET MVC 5**. Apress, 2013. 832 p.

FREEMAN**,** Eric; GREENE, Jennifer. **Use a Cabeça: C**#, Alta Books, 2008.

MACHADO, Leopoldinho; BEIGHLEY, Lynn. **Use a Cabeça: SQL**, Alta Books, 2008. 640 p.

BERNERS-LEE, Tim; MICROSOFT et al; **Request For Comments** 2616 Hypertext Transfer Protocol: HTTP/1.1. [Internet] Junho de 1999 disponível em <HTTPS://www.ietf.org/rfc/rfc2616.txt> acesso em 04/03/2014 às 13:00.

BENNERS-LEE, Tim; JAFFE, Jeffrey. **Consórcio Internacional World Wide Web.**

BLUM, Richard. **C# Networking Programming** EUA: John Wiley & Sons, 2006. 672 p.

MICROSOFT. **Micrososft Developer Network (MSDN).**

TADASHI, Alexandre. **Introdução ao ASP.NET Razor**, MSDN: Janeiro/2011. 1 p.

CHADWICK, Jess. **Programming Razor**, O'Reilly Media, Inc, 2011. 120 p.

FOWLER, Martin. **Patterns of Enterprise Application Architecture**, 2003. 557 p.

BEAULIEU, Allan. **Learning SQL**, O'Reilly Media, Inc. 2009. 338 p.

ISO. International Organization for Standardization, disponível em: <HTTP://www.iso.org> acessado em 05/03/2014.

CASTRO, Elizabeth. **HTML**: **Visual QuickStart Guide,** Estados Unidos. Peachpit Press**: 2011.** 576 p.

MCFARLAND, David. **JavaScript & JQuery: The missing Manual**, O'Reilly Media, Inc, 2011. 518 p.

DeMarco, **Structured Analysis and System Specification**, 1979. [Princípios fundamentais do OOD]

ANDRADE, Antônio. **Usabilidade de interfaces Web**: Avaliação heurística no jornalismo on-line, E-papers. 142 p.

ANDERSON, Rick. **Bundling e minificação**, disponível em <http://www.asp.net/mvc/tutorials/mvc-4/bundling-and-minification>, acessado em 20/03/2014 às 23h00min.

WASSERMAN, Stanley; FAUST, Katherine. **Social Network Analysis**: Methods and Applications, Cambridge Univesity Press, 2009. 825 p.

RYAN, Peter. **Social Netowrking**, The Rosen Publishing Group, 2011. 48 p.

MARTIN, Robert. **The single responsibility principle**, 2011.

W3C (Estados Unidos). World Wide Web Consortium (Org.). **CSS selectors**. 2014. Disponível em: <http://www.w3.org/TR/selectors/#class-html>. Acesso em: 15 mar. 2014 às 21h00min.