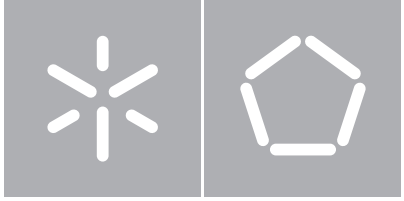
Universidade do Minho
Escola de Engenharia

João Pedro Pereira Peixoto

Análise e conceção de aplicações móveis de apoio ao processo educativo

Julho de 2015



Universidade do Minho
Escola de Engenharia
Departamento de Informática

João Pedro Pereira Peixoto

Análise e conceção de aplicações móveis de apoio ao processo educativo

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor António Manuel Nestor Ribeiro
Professor Nuno Miguel Feixa Rodrigues

Julho de 2015

AGRADECIMENTOS

Esta tese de mestrado reúne o estímulo e o apoio de instituições e de várias pessoas aos quais estou grato.

Ao meu orientador, António Nestor Ribeiro, que sempre se mostrou disponível e me apoiou mesmo nas decisões mais complicadas. Ao meu co-orientador, Nuno Feixa Rodrigues, com quem, inicialmente, discutia com regularidade o caminho a tomar.

À Codevision pela oportunidade e a todos os colaboradores que integram esta empresa, pois acolheram-me num ambiente fantástico.

À Universidade do Minho e aos docentes que fizeram parte desta jornada, pela formação de excelência. Aos meus colegas que me acompanharam e apoiaram nesta viagem.

Em especial, à minha família que me concedeu esta oportunidade e sempre me inspirou ao longo deste percurso.

ABSTRACT

Web development evolution and more capable devices have brought higher boundaries, being able to develop web applications with much more complexity and regarding that more different devices with different characteristics are used to access these applications.

This evolution has demanded web development framework companies to evolve their products and to reduce their product cycle.

Codevision, which always had their products on a web architecture, feels compelled to keep up with this evolution.

This dissertation should come up with two different middlewares with all the latest of web development technologies in order to make two applications to satisfy the needs regarding Codevision.

Reading all the steps and decision makings should help any developer with ASP.NET MVC4 and Windows 8 applications. It also should give them a better understanding of applications architecture and the need to think ahead making a solid middleware for further achievements. Some frameworks will be used in order to ease user experience and to ease developer's work.

RESUMO

Com a evolução do web development e dos dispositivos que acedem as estas aplicações, os limites de hardware e conectividade que anteriormente existiam foram atenuados, tornando possível o desenvolvimento de aplicações muito mais complexas e que têm de ter em atenção o maior universo de dispositivos e diferentes características, que são utilizados para aceder a estas.

Tendo em conta esta evolução, as empresas que disponibilizam frameworks para web development têm progredido na conceção dos seus produtos, apresentando mais regularmente novas soluções.

Neste sentido, a Codevision, empresa que sempre optou por ter os seus produtos numa arquitetura web, considera primordial acompanhar esta evolução.

Assim, esta dissertação propõe-se em desenvolver dois middlewares que tirem vantagem das novas funcionalidades web para serem utilizados no desenvolvimento de duas aplicações que vão de encontro a esta evolução, de modo a satisfazer carências da empresa.

O detalhar deste desenvolvimento deverá servir como uma espécie de tutorial para o desenvolvimento de aplicações ASP.NET MVC4 e de Windows 8 apps. Além disso, deverá focar-se na importância da escolha de uma boa arquitetura e no desenvolvimento de um middleware sólido, contribuindo para um desenvolvimento mais fácil e impulsionando a sua evolução. Serão utilizadas algumas frameworks para ultrapassar alguns obstáculos relativamente à usabilidade do utilizador e para facilitar a concretização dos objetivos por parte do programador.

CONTEÚDO

LISTA DE FIGURAS

EXEMPLOS DE CÓDIGO

Parte I

MATERIAL INTRODUTÓRIO

1

INTRODUÇÃO

1.1 CONTEXTO

O mercado de tablets tem crescido bastante nos últimos anos (Gartner, 2014) como se comprova na Figura 1, que apresenta um total de, aproximadamente, 116 milhões de tablets vendidos em 2012, e na Figura 2, com, aproximadamente, 195 milhões de tablets vendidos em 2013. Estas figuras também demonstram que não existe uma tendência para apenas um sistema operativo. Este crescimento fez com que as vendas de computadores portáteis abrandassem, havendo cada vez mais pessoas a optarem pelos tablets em detrimento dos portáteis. Existem muitas discussões quanto à capacidade de um tablet substituir um portátil, mas, pelo menos, do ponto de vista das operações básicas do dia-a-dia, este é altamente eficaz, e até se podem considerar algumas vantagens como a portabilidade e a durabilidade da bateria.

Com este aumento de vendas, surgiu a necessidade das empresas adaptarem o seu software, de modo a este ser usável nestas plataformas e até a tirarem partido de algumas funcionalidades que estes introduziram, tornando as aplicações mais atrativas para o utilizador. Assim, surge um problema para as empresas, pois têm de contratar programadores especializados ou deslocar programadores da sua área para o desenvolvimento das aplicações móveis. Estas podem trazer um obstáculo acrescido, pois cada plataforma tem a sua linguagem de programação.

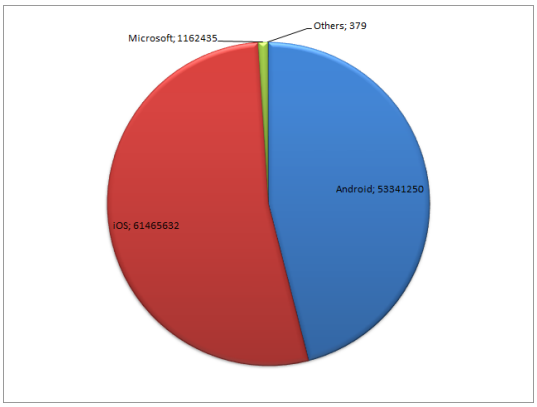


Figura 1.: Venda de tablets em 2012

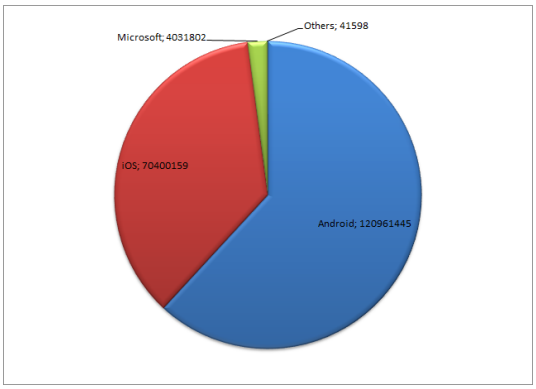


Figura 2.: Venda de tablets em 2013

A Codevision, empresa de software responsável pelos casos de estudo que serão desenvolvidos ao longo desta dissertação, pretende que sejam feitas duas aplicações tendo em mente a usabilidade no tablet, com especial atenção para a sua apresentação e manutenção.

Para o desenvolvimento destas aplicações serão utilizadas tecnologias emergentes, tanto para a arquitetura destas como para muitas das bibliotecas importadas. Nota-se uma grande preocupação das grandes empresas em facilitar o processo de desenvolvimento para aplicações *mobile ready*, assim como em dar apoio ao desenvolvimento de bibliotecas, que facilitem o trabalho dos programadores. Devido à grande necessidade neste tipo de desenvolvimento, estas estão em constante atualização, o que significa que estão a ser cada vez mais utilizadas.

Tal como foi referido acima, o aumento da utilização de tablets é uma motivação, pois significa que as aplicações irão atingir um público-alvo maior e em expansão, público-alvo que está relativamente definido e que maioritariamente se tornará utilizador das aplicações lançadas pela empresa, pois facilitar-lhe-á o seu trabalho ou será um complemento às aplicações que já possuem.

Os dois casos de estudo apresentados pela Codevision, são dois projetos distintos, mas ambos partilham da necessidade de modernização e adaptação ao panorama atual da proliferação do uso de tablets.

Um dos projetos é um complemento a um produto já existente, o E-Schooling Server, um sistema de informação que permite às instituições de ensino gerir toda a informação escolar através de diversos perfis e funcionalidades. Este projeto focar-se-á nos perfis de docente e diretor de turma, permitindo-lhes realizar as mesmas operações mais facilmente, com algumas novas funcionalidades e com suporte para uso de tablet. Este projeto surgiu da necessidade dos docentes realizarem as operações básicas que o E-Schooling Server, fornece sem terem de recorrer a um computador. O desenvolvimento deste projeto terá de ter em atenção uma futura expansão dos perfis e, consequentemente, de funcionalidades, tal como outros aspetos, nomeadamente a internacionalização.

A facilidade com que os alunos têm acesso a um tablet, principalmente por causa do programa que a Microsoft implementou em Portugal, com a distribuição de Surfaces a um preço reduzido nas escolas, motivou o outro projeto apresentado, que tem como objetivo ser um complemento para os alunos de ajuda aos conteúdos lecionados. Pretende-se uma aplicação em que estes acedam de forma fácil à informação e que promova a melhoria das aprendizagens. Esta aplicação irá consumir informação gerada no E-Schooling Server para fornecer dados e materiais relevantes aos alunos.

1.2 OBJETIVOS

As duas aplicações, referidas na secção, anterior levarão ao estudo da melhor arquitetura a utilizar, pois esta definirá a sua futura expansão. Uma arquitetura estruturada e pensada no tipo de aplicação que se pretende desenvolver é muito importante, bem como deverá ter em conta as necessidades presentes e futuras.

A framework de desenvolvimento deverá possibilitar a extensibilidade. Esta característica é muito importante, pois um dos objetivos mais importantes deste trabalho é criar uma base sólida de desenvolvimento (middleware), sendo assim facilitado o processo de desenvolvimento dos futuros programadores da aplicação, não tendo, então, de reescrever o mesmo código múltiplas vezes. Esta base deverá ter em atenção vários aspetos: a necessidade de controlos especiais para a interface do utilizador, a internacionalização das aplicações, a possibilidade de configurar permissões consoante o perfil do utilizador, ou até mesmo do utilizador em si, entre outros.

No middleware anteriormente referido, será importante explorar algumas bibliotecas e estendê-las criando, assim, novas possibilidades de utilização por parte do programador, viabilizando desta forma novas funcionalidades para o utilizador.

Deverão ser consideradas várias arquiteturas, frameworks e bibliotecas para se optar pelas que melhor se adaptam ao que se pretende. Para fazer esta escolha deve-se ter em atenção os objetivos referidos anteriormente, assim como as funcionalidades que as aplicações terão de ter e que serão apresentadas mais à frente.

Estas escolhas serão aplicadas no desenvolvimento das duas aplicações mencionadas anteriormente. A primeira aplicação, a aplicação que servirá como complemento ao E-Schooling Server, e que tem como ponto de partida a focagem nos perfis de docente e diretor de turma, chamar-se-á FlatMountain. A outra aplicação, a aplicação para complementar a aprendizagem dos alunos na sala de aula, chamar-se-á Win8 app.

No final desta dissertação é expectável ter duas aplicações funcionais com todos os requisitos mencionados. Estas deverão estar preparadas para uma fácil expansão de funcionalidades, por parte de outros programadores e deverão possibilitar a configuração de vários aspetos, por parte do administrador destas.

1.3 ESTRUTURA DO DOCUMENTO

Na introdução é apresentado o contexto do problema assim como os objetivos gerais a cumprir. Seguidamente, é feito um enquadramento tecnológico, expondo arquiteturas, frameworks e bibliotecas que poderão ser utilizadas no projeto. Posteriormente, será apresentado o problema com mais detalhe e o que se espera obter até ao final deste trabalho. Com o estudo feito e o problema analisado, no capítulo do desenvolvimento apresentam-se as várias estratégias adotadas para ultrapassar os problemas encontrados e como se atingiram os objetivos. As aplicações feitas são apresentadas no capítulo das aplicações. Finalmente, na conclusão e trabalho futuro reflete-se sobre todo o trabalho realizado, o cumprimento dos objetivos e aquilo que se perspetiva para o produto.

2

ENQUADRAMENTO TECNOLÓGICO

No presente capítulo irão ser apresentadas arquiteturas possíveis para o desenvolvimento de aplicações web, frameworks que definem e auxiliam este desenvolvimento e algumas bibliotecas que se consideram importantes para dar profundidade às aplicações.

2.1 ARQUITETURAS

Esta secção pretende abordar as diferenças entre a arquitetura server-side web, uma arquitetura mais usada no passado, e a arquitetura client-side web, uma arquitetura que tem vindo a ganhar terreno face à anterior.

2.1.1 *Server-side web*

Este tipo de arquitetura utiliza o browser do lado do cliente apenas para fazer o render da página. Toda a computação é feita do lado do servidor (Figura 3).

Assim, sempre que o utilizador pretende fazer uma operação, tem de ser feito um pedido ao servidor, que executa as operações necessárias e envia uma nova página para o cliente.

Este tipo de abordagem está sempre dependente da ligação à internet, havendo muitas ligações entre o cliente e o servidor, sobrecarregando o servidor, pois, se tiver muitos clientes para atender, tem muitos pedidos para processar. Este problema também prejudica o utilizador porque, geralmente, o atraso que a ligação provoca é maior do que o tempo que demoraria a processar o pedido localmente.

Esta arquitetura fazia sentido quando as máquinas pessoais não tinham grande capacidade de processamento.

2.1.2 *Client-side web*

Esta arquitetura retira alguma da computação feita do lado do servidor, passando esta a existir também do lado do cliente. Ao contrário de arquiteturas como *server-side web*, esta já contempla código a executar em *background*. Este tipo de comportamento geralmente é atingido usando JavaScript.

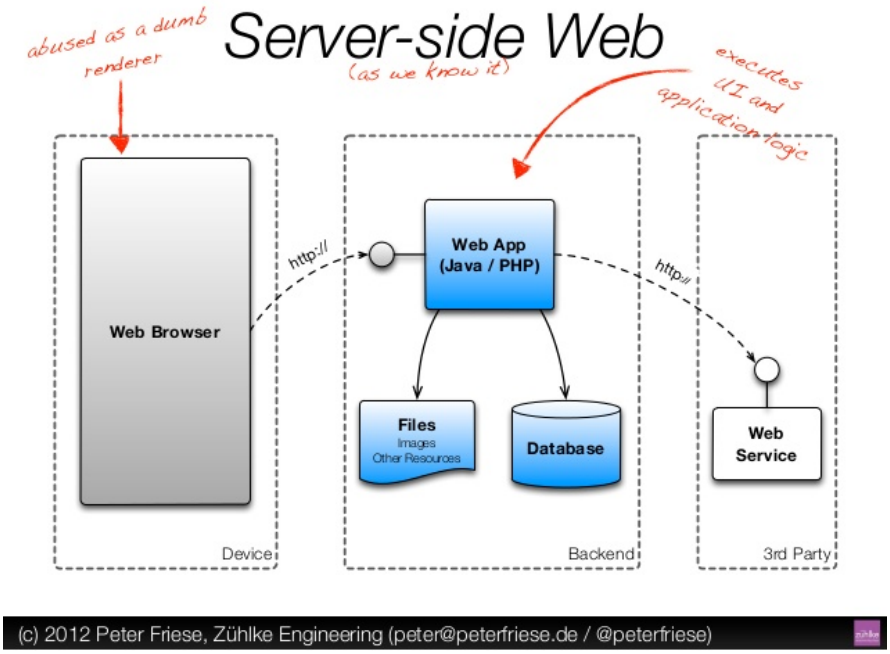


Figura 3.: Server-side web arquitecture

Com este tipo de computação, do lado do cliente, alcança-se maior rapidez, pois não tem de se estar constantemente a realizar pedidos ao servidor, que, mesmo em operações simples, tem sempre o problema da latência da ligação. Para atenuar esta latência aos olhos do utilizador, os pedidos podem ser assíncronos. Utilizando as novas características do HyperText Markup Language (HTML), apresentadas anteriormente, estes pedidos ainda serão menos constantes.

A Figura 4 é bastante ilustrativa no que se refere a este comportamento, indicando que, do lado do cliente, para além da interface gráfica, também existe computação e dados aos quais a aplicação pode aceder, sem necessidade de realizar comunicação com o servidor.

Uma aplicação bastante conhecida e que tirou partido deste tipo de arquitetura, tornando-se um exemplo, foi o *GMail* (Weinstein, 2012). Este, com o seu aparecimento tardio comparativamente com outros serviços de *webmail* já existentes e bastante populares, tal como o *Hotmail*, teve de revolucionar de alguma forma e a adopção desta arquitetura foi uma das características para o seu sucesso de hoje.

2.2 FRAMEWORKS

Frameworks são plataformas para o desenvolvimento de aplicações. Esta fornece uma base para o programador fazer a aplicação e está associada a uma certa plataforma. Geralmente, uma framework contém classes, funções, uma application programming interface (API), bibliotecas, um compilador,

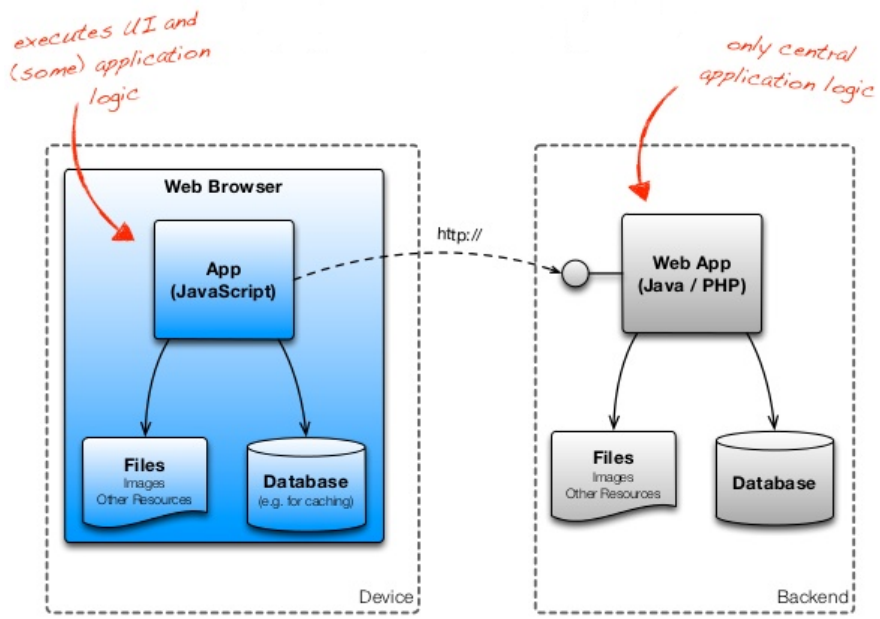


Figura 4.: Client-side web arquitecture

entre outros, que ajudam o programador a alcançar o seu objetivo, sem ser necessário reinventar muitos destes aspetos que são utilizados em muitas aplicações desenvolvidas.[1]

Nesta secção serão apresentadas algumas frameworks utilizadas.

2.2.1 *ASP.NET MVC (Galloway et al., 2012)*

A framework Model-View-Controller separa a aplicação em três componentes principais: o modelo, a view e o controller.

O modelo é uma classe que representa o domínio no qual estamos interessados. É aqui que a aplicação comunica com as camadas inferiores, tanto para povoar os objetos, como para realizar métodos específicos da camada de negócio do domínio no qual está inserido, até à persistência dos dados.

A view é onde é gerado o HTML dinamicamente para apresentação dos dados.

O controller é que trata da relação entre o modelo e a view. Recebe pedidos do utilizador, comunica com o modelo e serve-o à view (Figura 5).

Este tipo de arquitetura facilita o programador na separação da lógica da sua aplicação, definindo muito bem onde cada tipo de lógica se deve situar na aplicação. A lógica da interface do utilizador deve ser colocada na view, a lógica de pedidos no controller e a lógica de negócio deve ser colocada no modelo (Team, 2009).

1 Retirado de http://techterms.com/definition/framework

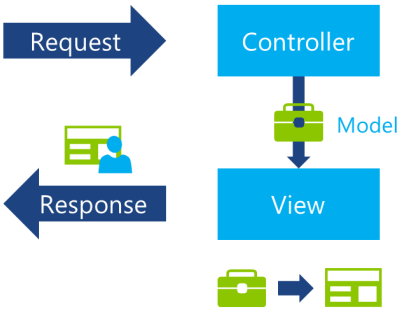


Figura 5.: MVC

2.2.2 *Windows 8 App - Javascript model*

Com o lançamento do Windows 8 e com este sistema operativo a ser usado tanto em computadores como em tablets, a Microsoft lançou uma framework para os programadores poderem criar aplicações para este sistema, sem terem de atravessar uma grande curva de aprendizagem, caso fossem web developers. Para desenvolver neste paradigma, apenas é necessário saber HTML, Cascading Style Sheets (CSS) e Javascript, as linguagens básicas de qualquer aplicação web.

Neste tipo de modelo, para alimentar a aplicação em termos de dados, estes podem estar guardados localmente (imagens fixas, ficheiros, entre outros) ou utiliza-se um qualquer webservice e acede-se a este, por exemplo, com uma chamada AJAX[2].

2.2.3 *Entity Framework*

Esta framework é uma object-relational mapping (ORM). Este tipo de frameworks permite fazer o mapeamento de bases de dados relacionais, representando-as em objectos.

Com esta framework, o acesso a entidades é instantâneo, não sendo necessário a escrita de queries em SQL. Com o uso de language-integrated query (Linq) e a capacidade de fazer lambda expressions, consegue-se realizar uma grande parte das queries necessárias, com muito menos esforço e bastante menos código.

2 Asynchronous Javascript and XML, é um grupo de técnicas no desenvolvimento web que possibilita o envio e recebimento de dados para/do servidor assincronamente

2.2.4 *Windows Communication Foundation*

Windows Communication Foundation (WCF) é uma framework que possibilita a criação de aplicações *service-oriented*. Com o WCF é possível enviar dados de um ponto para outro assincronamente. Este serviço pode correr em Internet Information Services (IIS), ou seja, tal como todas as web applications criadas em tecnologias Microsoft.

Assim, uma outra aplicação será o ponto que faz os pedidos a este serviço. Este responde com uma mensagem que pode ser em vários formatos. O mais interessante quando se cria uma aplicação web, talvez seja o formato JavaScript Object Notation (JSON).

2.3 RESPONSIVE WEB DESIGN (MARCOTTE (2010))

É um tipo de web design focado para o utilizador ter uma melhor experiência de navegação, conforme as características do dispositivo utilizado para aceder ao conteúdo.

Esta abordagem é caracterizada pelas grids fluídas, imagens flexíveis e media queries no CSS. Para tornar as grids fluídas, o tamanho dos elementos da página devem ser sempre relativos, utilizando unidades como percentagem ao invés de valores absolutos. As imagens flexíveis são imagens cujo tamanho não é fixo, mas também relativo ao elemento onde se encontram. As media queries do CSS são regras que se podem definir para utilizar diferentes estilos de CSS, conforme as características do dispositivo utilizado. A regra mais utilizada é a largura do ecrã do dispositivo.

2.4 PRINCIPAIS BIBLIOTECAS

Uma biblioteca consiste em código já compilado e que serve um propósito bem definido. Esta contém classes, funções, scripts, entre outros, que ajudam o programador não tendo este de programar esta camada inferior. Um exemplo bastante simples é uma biblioteca de matemática, quando é necessário realizar qualquer operação matemática, não tendo estas de serem programadas.[3]

2.4.1 *ASP.NET SignalR*

É uma biblioteca que permite criar funcionalidades, que executam em tempo real numa página web, ou seja, é possível enviar informação para a página web, quando algo acontece do lado do servidor, sem que o utilizador tenha de fazer refresh na página para verificar as alterações.

Esta biblioteca utiliza WebSockets do HTML5 quando possível, ou tenta utilizar outras estratégias, sem que o utilizador, ou até mesmo o programador, se aperceba disso, pois o código mantém-se igual.

3 Retirado de http://www.techopedia.com/definition/3828/software-library

Funcionalidades, tais como chats ou gráficos em constante atualização, tornam-se muito mais simples de implementar e a experiência do utilizador é bastante melhorada.

2.4.2 *DataTables*

DataTables é um plugin de geração de tabelas para o jQuery.

As tabelas são uma forma bastante apelativa para apresentar grandes quantidades de dados. Este plugin facilita a criação da tabela, tem inúmeras opções e tem suporte por parte da comunidade.

Uma das grandes vantagens deste plugin, para além de ser open-source, ou seja, permitir uma costumização por parte do programador, é a possibilidade de a tornar server-side. Esta opção é muito importante quando se lida com uma grande quantidade de dados, pois para além de não sobrecarregar a página do cliente, as pesquisas suportadas, que podem ir ao nível da coluna, e atos como paginação, tornam-se bastante mais rápidos.

Outras opções importantes são a possibilidade de esconder colunas e a possibilidade de ordenação por coluna.

Como esta é uma biblioteca bastante usada, já existem pacotes que facilitam a sua usabilidade programaticamente na framework ASP.NET MVC tirando partido de alguns conceitos desta framework, evitando assim código repetido.

2.4.3 *FullCalendar*

O FullCalendar é um plugin para o jQuery que permite criar um calendário de eventos e que fornece muitas opções.

Este calendário é baseado no Google Calendar. A sua application programming interface (API) é bastante extensa permitindo ações tais como *drag and drop*, tratamento de ações como *onClick*, *onHover*, entre outras.

A apresentação deste calendário também permite bastantes opções. É possível visualizá-lo em formato de agenda semanal ou diária, ou em formato mensal. Também possibilita, à partida, a criação de eventos com diferentes *CSS classes*, o que faculta uma diferenciação do tipo de eventos pertencentes ao calendário.

2.4.4 *Flot*

Flot é uma biblioteca que permite o desenho de vários tipos de gráficos para o jQuery.

Para além de uma grande miscelânea de tipos de gráficos disponíveis, tem também uma extensa API que possibilita tratar ações, tais como o *onClick*. Além disso, no campo da costumização, também é muito versátil, pois os dados que este recebe já estão preparados para receber propriedades, como, por exemplo, a cor.

Uma possibilidade de fornecimento de dados, que pode ser interessante é, através de ajax, fazer *poll* de dados que estejam constantemente a ser atualizados.

3

O PROBLEMA

Tal como foi referido na introdução, existem dois produtos que a Codevision pretende lançar focados em tecnologias móveis. Uma das aplicações será algo complexa e deverá ter atenção a muitos aspetos e terá uma arquitetura multi-camada, o FlatMountain. A outra aplicação será mais simples e servirá como caso de estudo bem como poderá evoluir para algo que entrará no mercado, a aplicação Win8 app.

Para desenvolver o FlatMountain, antes de se evoluir para o acrescentar de novas funcionalidades ao utilizador, deve ser feita uma estrutura base, ou seja, uma camada entre a framework utilizada para o seu desenvolvimento e aquela que é realmente a programação feita para realizar as funcionalidades pretendidas. Neste sentido, e numa primeira fase, deverão ser desenvolvidos novos métodos de criação de controlos com reutilização de código; uma base para a configuração de permissões que abranja toda a plataforma; um sistema de traduções fácil de utilizar e que não tenha de envolver o programador, quando se pretende introduzir uma nova língua; controlos que suportem grande escalabilidade; novos controlos baseados em controlos existentes e em bibliotecas utilizadas para satisfazer diferentes tipos de interações e uma funcionalidade que seja em tempo real para mais tarde explorar para que outras possam tirar partido desta.

Depois de estes requisitos estarem cumpridos, ou seja, apenas tendo uma estrutura sólida, de fácil manutenção e facilmente escalável é que se deve partir para os requisitos das funcionalidades dos utilizadores da aplicação. Assim, pretende-se facilitar o trabalho futuro dos programadores desta aplicação.

As secções seguintes descrevem o produto pretendido, as funcionalidades a implementar e os principais desafios na implementação destas.

3.1 FLATMOUNTAIN

O E-Schooling Server está desenvolvido em ASP.NET *classic*, tendo um comportamento indesejável em termos de experiência de utilização, pois este abre página sobre página, aquando da sua navegação, tornando a experiência do utilizador bastante maçuda. Este é um dos aspetos principais a

alterar no desenvolvimento da nova aplicação, ou seja, esta deve ser *flat*, utilizando apenas algumas *modals*[1], quando estas se enquadrem na operação a realizar.

Outra grande mudança que deve ser feita é o aspeto. O E-Schooling Server não tem um aspeto amigável para a utilização com toque, nem tem um design *responsive*, havendo operações impossíveis de realizar e elementos não visíveis em certas resoluções.

Tal como no E-Schooling Server, esta aplicação terá a necessidade de apresentar bastante informação em forma de listagens. Estas listagens, por vezes, são pesadas e devem ter a sua computação em *server-side*, ou seja, ao cliente só devem chegar os dados relativos à página que este está a visualizar. Em qualquer filtragem ou mudança de página por parte do utilizador deve ser feito um pedido ao servidor, para este responder com os dados a apresentar.

Devem, então, ser pensadas novas formas mais apelativas de apresentar alguns dos dados, tal como a utilização de gráficos.

Outro aspeto relevante, e que se deve ter em conta à partida, é a internacionalização. Nenhum texto deve estar fixo na página. As datas e caixas com dados do tipo "dinheiro"também devem estar conforme a cultura. Esta adaptação da página não deve ser conforme a cultura do browser do utilizador, mas conforme a cultura escolhida pela administração do sistema no E-Shooling Server.

O sistema de mensagens deve ser intuitivo e personalizável. A troca de mensagens deve ser instantânea e a procura de utilizadores da aplicação fácil de realizar. Estas devem produzir notificações, caso o utilizador não esteja na página de chat com essa pessoa. A constituição de grupos para chat em grupo deve ser possível e de fácil manutenção.

Para esta fase da aplicação, pretende-se disponibilizar as operações que os docentes e diretores de turma possuem no E-Schooling Server, ou seja:

- Sumariar
- Marcar faltas
- Consultar a ficha de aluno
- Lançar notas
- Consultar as suas faltas possibilitando pedidos de justificação
- Comunicar com os restantes utilizadores da plataforma

3.2 WIN8 APP

A Codevision já detém um produto, o E-schooling Portal, que é um portal online onde os alunos e encarregados de educação podem consultar informações tais como o horário, as classificações, os movimento financeiros, entre outras.

1 Modals são janelas filhas da janela com que o utilizador está a interagir. Estas são apresentadas quando é necessário fazer alguma ação que não se consegue realizar na janela principal, não quebrando assim o fluxo de trabalho.

3.2. Win8 app

Com esta nova aplicação pretende-se criar algo mais interativo para o aluno, sendo até uma ferramenta de uso na sala de aula e de complemento à sua aprendizagem.

Será criada uma aplicação com o intuito de ser utilizada em Surface, onde o aluno poderá fazer as seguintes operações:

- Consultar horário
- Consultar turma
- Consultar disciplinas, classificações e faltas
- Aceder aos sumários das disciplinas
- Em cada sumário, poder descarregar anexos que o professor tenha colocado
- Poder tirar notas por sumário
- Disponibilizar as suas notas ou adicionar ficheiros
- Poder aceder às notas e ficheiros partilhados pelos colegas

Sendo uma aplicação para Windows 8, devem-se manter alguns aspetos em termos de usabilidade e design. A aplicação deve alongar-se horizontalmente e não verticalmente como é usual. Os ícones, caixas e os restantes elementos devem manter o estilo metro, estilo que caracteriza o Windows 8 e que se reconhece pelas formas retas, cores simples e sólidas.

Antes de adotar uma framework para o desenvolvimento, deve-se pensar que poderá ter de surgir, futuramente, a migração desta aplicação para outras plataformas, tais como Android ou iOS. Neste sentido, os dados devem ser fornecidos por um web service e a aplicação utilizar linguagens universais, que sejam de fácil adaptação.

Parte II

NÚCLEO DA DISSERTAÇÃO

4

DESENVOLVIMENTO

Este capítulo focar-se-á principalmente no desenvolvimento da base, que se referiu no capítulo do problema (capítulo 3). Pretende-se demonstrar a arquitetura das diferentes aplicações, assim como as principais funcionalidades, principalmente aquelas que são gerais a qualquer aplicação do mesmo tipo.

Aqui será feita uma demonstração de como se estende a classe HtmlHelper da framework MVC para se fazer os próprios controlos, sendo estes reutilizáveis em qualquer página da plataforma. Além disso, demonstra-se como estender a biblioteca Datatables, introduzindo assim novas funcionalidades como a introdução de ações no controlo e a transformação do controlo numa lookup[1]. Apresenta-se, também, o desenvolvimento do sistema de permissões, que controla o que cada perfil/utilizador pode ver/editar em cada área de edição, assim como que ações pode realizar e que tabelas de dados pode ver.

4.1 FLATMOUNTAIN

Inicialmente o desenvolvimento desta aplicação estava pensado para ser puramente web (HTML + JavaScript + CSS) com os dados a serem fornecidos por um web service. Isto permitiria tentar explorar frameworks, tais como o PhoneGap. Com o uso desta framework, seria possível gerar aplicações instaláveis e que se poderiam colocar nas diferentes lojas.

Esta abordagem foi abandonada, quando se decidiu que futuramente esta aplicação poderia ser expandida a mais perfis de utilizadores e não apenas aos inicialmente propostos: professor e diretor de turma. Com a análise desta expansão, e conhecendo o E-Schooling Server, o aumento da lógica na visualização de conteúdos e ações fez com que se mudasse o rumo para uma abordagem que tivesse *code-behind*, ou seja, que por trás da camada de apresentação tivesse a possibilidade de implementar lógica.

Assim, tendo presente que a Codevision desenvolve os seus produtos em tecnologias Microsoft e que esta tem apostado no desenvolvimento web com a evolução da sua framework ASP.NET MVC,

1 Lookup é um controlo utilizado para escolha de um objeto de um dicionário, tal como as DropDownLists, mas onde o dicionário é demasiado grande para ser utilizada uma DropDownList.

optou-se por utilizar esta framework. A versão mais recente desta era a quatro, e foi com esta que se avançou. Entretanto já saiu a versão cinco.

4.1.1 *Arquitetura*

Para existir uma boa separação de conceitos, dado que a aplicação pode evoluir para algo bastante grande, decidiu-se utilizar dois padrões arquiteturais: MVC e n-tier. Tal como se pode ver na Figura 6, o padrão MVC será usado como camada de apresentação e terá um pouco da lógica de negócio. A camada de negócio será repartida entre uma Business Layer Logic (BLL) e o modelo do MVC. A camada de dados é feita com um ORM criado em Entity Framework, que acede a uma base de dados Sql Server.

No projeto MVC, a estrutura deve manter-se, criando apenas algumas pastas para a introdução de conceitos necessários, como se vai demonstrar mais à frente. Este tipo de projeto já separa os Modelos, as Views, os Controllers e conteúdo, tal como imagens, bibliotecas JavaScript e Stylesheets.

No outro projeto, criam-se os modelos Entity Framework. Estes devem estar separados por área, pois um modelo não deve ter demasiadas entidades, uma vez que torna o sistema mais lento (Marchant, 2009). Para cada modelo, cria-se uma classe que servirá de BLL.

Com esta estrutura, a comunicação ideal entre as partes será o utilizador fazer uma chamada, que será tratada pelo controller. Este chama o modelo pretendido que, se tiver de comunicar com a base de dados, fá-lo-á através da BLL, correspondente à área pretendida. Quando o modelo tiver a informação necessária, o controller envia o modelo para a view e o resultado é apresentado ao utilizador.

Existe um outro projeto na solução que está omitido na Figura 6, que é o projeto que terá as Resources necessárias para a internacionalização da aplicação. A decisão de as colocar num projeto à parte, e não no projeto da camada de apresentação, partiu da ideia que um conjunto de Resources é basicamente um dicionário, com diferentes traduções e, então, futuramente, poderá ser utilizado noutras aplicações que partilhem os mesmos conceitos.

Outros projetos poderão ser importados, tais como projetos de bibliotecas, que até podem ser alterados, mas deverão ser importados como referências, tornando-os agnósticos à aplicação, para poderem ser utilizados noutras aplicações.

4.1.2 *ASP.NET MVC 4*

Como foi referido anteriormente, o fluxo de acontecimentos no MVC é o utilizador invocar algo na view, que é tratado pelo controller. Este chama o modelo apropriado, envia-o para view e esta é gerada e apresentada ao utilizador.

Um modelo deve conter as propriedades necessárias a apresentar na view. Tome-se como exemplo uma versão simplificada do modelo de um sumário.

4.1. FlatMountain

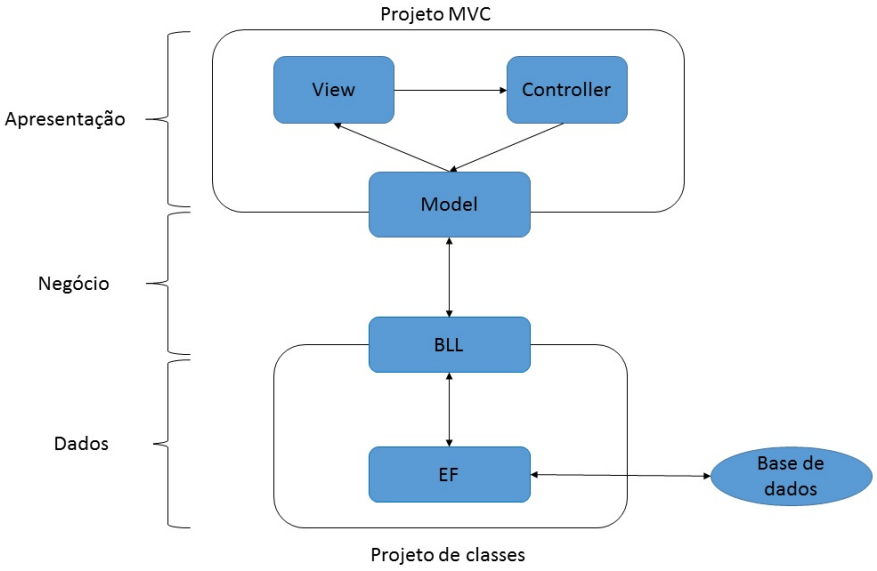


Figura 6.: Arquitetura do FlatMountain

```
public class SummaryModel
{
    [ScaffoldColumn(false)]
    public Guid SummaryId { get; set; }

    [ReadOnly(true)]
    [Display(Name = "Teacher", ResourceType = typeof(Resources.Resources))]
    public string TeacherName { get; set; }

    [ReadOnly(true)]
    [Display(Name = "Class", ResourceType = typeof(Resources.Resources))]
    public string ClassName { get; set; }

    [ReadOnly(true)]
    [Display(Name = "LessonNumber", ResourceType = typeof(Resources.Resources))]
    public int? LessonNumber { get; set; }

    [Display(Name = "Summary", ResourceType = typeof(Resources.Resources))]
    public string SummaryText { get; set; }
}
```

Neste modelo identificam-se cinco propriedades. Exatamente acima de cada propriedade temos DataAnnotations. Estas anotações servem para definir várias propriedades da propriedade em questão, tais como:

SCAFFOLDCOLUMN(FALSE) esta propriedade não é desenhada caso seja utilizado um sistema de scaffolding

READONLY(TRUE) o campo desenhado desta propriedade não irá ser editável

DISPLAY esta propriedade pode ter várias propriedades internas, tais como o nome do campo

STRINGLENGTH pode-se definir um tamanho máximo para um campo de texto. Este tipo de propriedades são bastante importantes pois, quando um utilizador submete um formulário, não é necessário tratar este tipo limitações do lado do servidor, uma vez que este é prontamente avisado do erro.

Quando um utilizador navega para uma página de um sumário, estas propriedades devem ser desenhadas. Para tal, a ação do controller deve estar definida para aceitar um id de um sumário, construir o modelo do sumário e enviá-lo para a view.

```
[HttpGet]
[MvcSiteMapNode(Title = "EditSummary", ParentKey = "Summaries")]
public ActionResult Edit(Guid summaryId)
{
    SummaryModel m = new SummaryModel(summaryId);
    return View(m);
}
```

Quando a view receber o modelo, esta deve desenhar as propriedades que este contém e que se pretendam apresentadas ao utilizador. Segue um simples formulário com algumas das propriedades do modelo apresentado anteriormente.

```
@using (Html.BeginForm())
{
  @Html.LabelFor(m => m.TeacherName)
  @Html.DisplayFor(m => m.TeacherName)

  @Html.LabelFor(m => m.LessonNumber)
  @Html.DisplayFor(m => m.LessonNumber)

  @Html.LabelFor(m => m.SessionNumber)
  @Html.TextBoxFor(m => m.SessionNumber)

  @Html.LabelFor(m => m.SummaryText)
  @Html.TextAreaFor(m => m.SummaryText)
  <div class="form-actions">
    <button type="submit" value="Save" class="btn btn-primary">@Resources.Save</
        button>
  </div>
}
```

Como a view utiliza Razor Syntax, pode-se introduzir código C#. A forma simplificada de desenhar etiquetas e caixas de texto apresentada é uma caraterística do ASP.NET MVC. Este tipo de métodos designam-se de HTML Helpers e vão ser discutidos posteriormente. Quando o utilizador submeter o formulário, este vai acionar a ação no controller do tipo POST. Este tipo de ação recebe um modelo do tipo SummaryModel com as alterações efetuadas pelo utilizador.

Validação de dados

Existem duas formas para validar os dados introduzidos pelo utilizador: por JavaScript e pelo estado do modelo após o mapeamento das propriedades preenchidas pelo utilizador.

Quando estas são validadas por JavaScript, o pedido ao servidor não chega a ser feito. O utilizador é instantaneamente alertado para os campos que contêm erros e com uma mensagem que identifica a razão do erro. Para esta verificação ficar ativa, é necessário colocar duas propriedades no ficheiro web.config com o valor a *true*: *ClientValidationEnabled* e *UnobtrusiveJavaScriptEnabled*.

Se a validação por JavaScript falhar, quando o cliente submete o formulário, na ação do tipo POST, deve-se verificar se o estado do modelo é válido. Juntamente com esta validação do tipo booleano, existe uma lista dos erros encontrados. Se o estado do modelo não for válido, devemos retornar à mesma View com o modelo. Este tem os erros que serão apresentados ao utilizador. Para tal deve-se preparar a página, caso este procedimento aconteça. Para facilitar a codificação, pode-se simplesmente utilizar um HTML Helper que apresenta a informação do erro na forma que se deseja.

4.1.3 *Design*

Como base para o design da aplicação foi usada a framework bootstrap. Esta é uma framework html, CSS e JavaScript responsive e com prioridade para dispositivos móveis. Esta opção é bastante utilizada pois facilita a criação de um site adaptativo a diferentes resoluções. Para tal são utilizadas regras, chamadas de *@media*, que definem novas regras de estilo conforme o dispositivo utilizado para aceder à aplicação. Os atributos mais utilizados nas media queries são:

- min-width
- max-width
- min-height
- max-height
- orientation=portrait
- orientation=landscape

No caso da aplicação em desenvolvimento, opta-se por esconder alguns elementos menos importantes, tais como a lista de utilizadores com os quais se pode falar na funcionalidade de chat, por adaptar o menu lateral, tornando-o mais pequeno e até colocá-lo no topo da página, e por reduzir o tamanho de alguns elementos e também o tamanho da letra utilizada.

Organizacionalmente, em termos de código, a framework ASP.NET MVC utiliza uma página como Layout, onde se podem invocar outras páginas, neste caso, por exemplo, o menu lateral ou a lista de utilizadores da funcionalidade de chat, e também é onde se define onde é colocada a página principal que vai ser alterada, conforme as Actions presentes nos Controllers.

4.1.4 *Chat com SignalR*

Um dos princípios do SignalR é ter uma classe que herda de Hub. Hub é uma classe que contém métodos que possibilitam a comunicação entre ligações SignalR.

Sendo assim, e no caso de um chat, na classe que herda de Hub, deve-se definir o método que trata o envio das mensagens às ligações SignalR desse Hub. Mas antes, para podermos comunicar com diferentes pessoas e grupos, ou seja, para a mensagem não ser partilhada entre todas as ligações a este Hub, deve-se fazer override ao método OnConnected e aproveitar uma característica do SignalR, os grupos.

```
public override Task OnConnected()
{
  // his own individual group. This allows others to send him messages
  string userName = Context.User.Identity.Name;
  Groups.Add(Context.ConnectionId, userName);

  // his groups so he receives messages sent to groups who he belongs
  BLLModelChat bllMC = new BLLModelChat();
  List<Guid> belongingGroups = bllMC.GetBelongingGroupsByUsername(userName);
  foreach (Guid group in belongingGroups)
    Groups.Add(Context.ConnectionId, group.ToString());

  return base.OnConnected();
}
```

Verifica-se o uso de um grupo por utilizador para a troca de mensagens individuais, mas também o uso de grupos no verdadeiro sentido da palavra, ou seja, os utilizadores do chat podem definir grupos de pessoas e trocar mensagens entre todos os seus elementos simultaneamente.

Para tratar as mensagens enviadas, o método deve receber uma mensagem, saber o seu remetente e destinatário e registar que métodos JavaScript irão fazer o seu tratamento do lado do cliente.

```
public void SendChatMessage(string from, string to, string message)
{
  // #discussion is identified by the userName of the person who's on the other
      side

  // get his userName to persist and check whether the message is sent to the
      other side
  string userName = Context.User.Identity.Name;

  // persist the message, get its ID and send this ID to the other side. If chat
      window open, use this ID to make message as read
  string[] mIds = persistMessages(to, message, userName);

  // check whether individual or group message. Set discussionId to use on the
      other side
  string discussionId = "";
  if (mIds.Length > 1)
    discussionId = to;
  else
    discussionId = userName;

  // send message to group
  Clients.Group(to, Context.ConnectionId).addNewMessageToOtherUserPage(from,
      discussionId, message, DateTime.Now.ToString("G"), mIds);

  // put the message on his chat window
  Clients.Caller.addNewMessageToMyPage(from, to, message, DateTime.Now.ToString("
      G"));
}
```

Como se pode constatar, este método tem mais lógica do que o simples envio das mensagens. Existe alguma lógica extra por causa da possibilidade de ser um grupo definido pelo utilizador ou um contacto individual deste.

Os métodos *addNewMessageToOtherUserPage* e *addNewMessageToMyPage* têm de estar definidos em JavaScript na página onde se fez a ligação. A ligação deve ser definida também no JavaScript onde se pretende utilizar esta funcionalidade.

```
// start chat
$(function () {
    // Reference the auto-generated proxy for the hub.
    var chat = $.connection.chatHub;

    // function to send message to other user
    chat.client.addNewMessageToOtherUserPage = function (name, group, message,
        time, messageIds) {
        processAddNewMessageToOtherUserPage(name, group, message, time,
            messageIds);
    };

    // function to put the message in his own page
    chat.client.addNewMessageToMyPage = function (name, group, message, time) {
        processAddNewMessageToMyPage(name, group, message, time);
    };

    startConnection(chat);
});
```

Os métodos *processAddNewMessageToOtherUserPage* e *processAddNewMessageToMyPage* vão simplesmente construir o html e colar na janela de chat. No caso de ser o método que trata a mensagem colada na janela do outro utilizador, como esta pode não estar aberta, apresenta uma notificação ao utilizador na área de notificações. Se a janela estiver aberta, para além de lhe ser apresentada a mensagem, esta é marcada como lida.

4.1.5 *HTML Helpers*

HTML Helpers são apenas métodos que retornam uma string. Estes podem ser chamados nas views devido à Razor Syntax com a notação "@". Muitos destes HTML Helpers são oferecidos à partida num projeto ASP.NET MVC 4.

A grande vantagem dos HTML Helpers é a simplificação do código HTML numa view e evitar que o programador tenha de escrever grandes quantidades de código e, por vezes, até repeti-lo inúmeras vezes para a representação de um elemento.

Assim, e se houver elementos que não são possíveis de desenhar com os HTML Helpers que vêm disponíveis por defeito num projeto ASP.NET MVC 4, o programador, se considerar que lhe será útil e irá utilizar múltiplas vezes esse elemento, pode criar o HTML Helper.

Adote-se o caso de estudo da definição de uma imagem em HTML. Para o projeto ficar organizado, deve-se criar uma pasta apenas para os HTML Helpers customizados, bem como uma simples classe e um método que devolverá uma string HTML.

```
public static IHtmlString Image(this HtmlHelper helper, string src, string
    CSSclass = null, string title = null, string alt = null, string align = null)
{
  TagBuilder img = new TagBuilder("img");
    img.Attributes.Add("src", VirtualPathUtility.ToAbsolute(src));
    img.Attributes.Add("class", CSSclass);
    img.Attributes.Add("title", title);
    img.Attributes.Add("alt", alt);
    img.Attributes.Add("align", align);
    return new MvcHtmlString(img.ToString(TagRenderMode.SelfClosing));
}
```

Neste exemplo verifica-se que podem ser definidos vários parâmetros opcionais que poderão ser atribuídos à imagem. Assim, para introduzir uma imagem numa view bastaria fazer algo como:

```
@Html.Image(@Url.Action("LoadPhoto", new { photoId = st.PhotoId }), "img-class-
    photo")
```

Um HTML Helper mais complexo, pois envolve Javascript e a propriedade de um modelo, seria para um controlo de escolha de data. Neste caso já temos de passar a propriedade do modelo e definir no Javascript, que corre no cliente, que esta entrada é do tipo *Datepicker* para ser utilizado o construtor do *Datepicker* do jQuery.

```
public static IHtmlString DatePicker<TModel, TValue>(this HtmlHelper<TModel>
    helper, Expression<Func<TModel, TValue>> expression)
{
  DateTimeFormatInfo format = Thread.CurrentThread.CurrentCulture.DateTimeFormat;

    TagBuilder div = new TagBuilder("div");
    TagBuilder input = new TagBuilder("input");
    ModelMetadata data = ModelMetadata.FromLambdaExpression(expression, helper.
        ViewData);
    string propertyName = data.PropertyName;
    DateTime? date = (DateTime?)data.Model;

    input.Attributes.Add("name", propertyName + ".Date");
    input.Attributes.Add("class", "grd-white");
    input.Attributes.Add("data-form", "datepicker");
    input.Attributes.Add("size", "16");
    input.Attributes.Add("type", "text");
    input.Attributes.Add("value", date.HasValue ? date.Value.ToString("d", format
        ) : null);
    input.Attributes.Add("dateFormat", format.ShortDatePattern);

    div.Attributes.Add("class", "input-append date");

    div.InnerHtml = input.ToString(TagRenderMode.Normal);

    return new MvcHtmlString(div.ToString(TagRenderMode.Normal));
}
```

O HTML Helper acima codificado pode ser chamado da seguinte forma:

```
@Html.DatePicker(m => m.InitDate)
```

Verifica-se que este está preparado para representar a data no formato da cultura definida e utiliza *reflection*[2] para ter acesso ao nome e valor da propriedade do modelo.

4.1.6 *Internacionalização*

Uma necessidade cada vez mais comum, quando se desenvolve uma plataforma web, é esta suportar diferentes línguas e regiões. Fala-se em regiões e não apenas em línguas, pois alguns países partilham

2 Reflection é usado no C# para aceder ao tipo de uma propriedade em tempo de execução. Com isto pode-se aceder aos seus campos e propriedades.

a mesma língua, mas não partilham o mesmo formato de datas, a mesma moeda, a mesma representação numérica, ... Como a internacionalização de uma aplicação pode ser algo complicado, deve-se tentar fazer algo de manutenção e desenvolvimento fáceis.

Uma abordagem possível é a de fazer uma View por cada língua, o que é bastante trabalhoso e difícil de manter. Então, a solução encontrada foi a utilização de resources. Foi criado um projeto apenas para guardar as resources. Resources são como um dicionário, onde se coloca uma chave, que deve existir em todas as línguas e o respetivo valor.

Como se pôde ver mais acima no modelo *StudentModel*, na anotação do nome a utilizar para cada propriedade, verifica-se o uso de Resources. Então, neste caso, o valor atribuído à propriedade *Name* é a chave da resource apontada pela propriedade *ResourceType*.

Como a intenção neste caso é a de forçar uma certa cultura conforme o produto, esta informação é guardada em base de dados e, aquando do acesso ao site, no método *Application_AcquireRequestState* no *Global.asax* regista-se a cultura pretendida. A partir deste momento todos os campos que apontem para uma chave das resources, o sistema vai tentar encontrá-la na resource respetiva da cultura registada. Caso a entrada não exista, procura-a na resource padrão.

Quando se pretende aceder às resources diretamente, pode-se utilizar o método *ResourceManager.GetString* com a chave como parâmetro.

No código JavaScript a solução passa por colocar num dicionário a resource a ser utilizada. Para tal cria-se uma View que vai buscar o ResourceSet, segundo a cultura utilizada, e copia-se esse ResourceSet para uma variável JavaScript. Esta View só é chamada nas páginas que necessitem de acesso às resources no código JavaScript.

O tratamento da região é tratado na construção dos HTML Helpers. Esta é necessária quando se utiliza um método HTML Helper para um DatePicker 4.1.5 ou quando se utiliza, p.e., um método HTML Helper para um campo de moeda.

```
public static IHtmlString Money<TModel, TValue>(this HtmlHelper<TModel> helper,
    Expression<Func<TModel, TValue>> expression)
{
  NumberFormatInfo format = Thread.CurrentThread.CurrentCulture.NumberFormat;

  ModelMetadata data = ModelMetadata.FromLambdaExpression(expression, helper.
      ViewData);
  string propertyName = data.PropertyName;
  decimal amount = (decimal)data.Model;

  TagBuilder div = new TagBuilder("div");
  TagBuilder span = new TagBuilder("span");
  TagBuilder input = new TagBuilder("input");

  div.Attributes.Add("class", "input-prepend");

  span.Attributes.Add("class", "add-on");
  span.SetInnerText(format.CurrencySymbol);

  input.Attributes.Add("name", propertyName);
  input.Attributes.Add("class", "grd-white");
  input.Attributes.Add("type", "text");
  input.Attributes.Add("value", amount.ToString(format));

  div.InnerHtml = span.ToString(TagRenderMode.Normal);
  div.InnerHtml += input.ToString(TagRenderMode.Normal);

  return new MvcHtmlString(div.ToString(TagRenderMode.Normal));
}
```

Mais uma vez se verifica a usabilidade dos HTML Helpers. Com estes, o programador pode gerar controlos que são dependentes da região, mas que tratam isto de uma forma transparente.

4.1.7 *DataTables*

A utilização desta biblioteca é bastante simples, pois apenas se tem de manter uma view, que irá ser reutilizada para todas as tabelas necessárias. Esta tem o HTML necessário para desenhar alguns dos seus elementos. Os restantes são gerados por JavaScript no seu construtor. Este construtor também faz parte desta página e consegue ser genérico, pois a página tem um modelo com as definições que cada tabela terá. Assim, numa página onde se pretende desenhar uma tabela, cria-se um objeto do tipo do

modelo desta, define-se as propriedades pretendidas, sendo obrigatório apenas definir a que método esta terá de ir buscar os dados do lado do servidor. Depois a view é chamada como uma Partial[3].

Estas conseguem ser genéricas, pois o método que devolve os dados é fortemente tipado, ou seja, sabe-se exatamente que tipo de objeto se está a retornar, uma vez que este tem de ser definido. Estes modelos de representação de dados das tabelas são muito parecidos com os modelos utilizados para a construção das páginas. Incluem propriedades e DataAnnotations 4.1.2.

O tratamento destes dados para a devolução do objeto de configuração da DataTable encontra-se num projeto à parte, importado no FlatMountain. Como necessidade de mais configurações disponíveis na DataTable, decidiu-se estender o objeto de configurações para acoplar estas configurações, mantendo assim intacto o projeto original.

Uma destas necessidades era a de haver uma ação disponível que mostrasse/escondesse os filtros por coluna nas DataTables, caso esta os tivesse. Para tal criou-se uma propriedade que é apenas uma *string*, que guarda a função necessária em JavaScript, para realizar tal ação para a DataTable específica, através do Id desta. No construtor da DataTable avalia-se se a filtragem por colunas está ativa para disponibilizar o botão para a ação ou não.

Uma outra opção é de a DataTable ser de um controlo do tipo Lookup. Quando tal acontece, é necessário que a DataTable, que está visível numa Modal, após o utilizador selecionar a linha pretendida para preencher o campo no qual a lookup foi ativada, se feche e preencha o campo com o texto escolhido e guarde o Id deste num campo escondido. O processo de programação destas ações pode-se tornar moroso pois o JavaScript é guardado como string nas propriedades do objeto. A.1

4.1.8 *Outras bibliotecas*

Apresentam-se de seguida outras bibliotecas utilizadas onde não foi necessário estender as suas funcionalidades, apenas foram feitas algumas adaptações.

FullCalendar

Esta biblioteca é bastante conhecida pois é de fácil uso e apresenta um calendário bastante parecido com o Google Calendar.

Para colocar um calendário com eventos numa página, na View basta pôr uma div onde será construído o calendário, via JavaScript. No código JavaScript, e para não atrasar o carregamento da página devido ao calendário, faz-se uma chamada ajax à Action que devolve os eventos em JSON a carregar no calendário. Em caso de sucesso da chamada ajax, chama-se o construtor do fullCalendar na div anteriormente referida, onde se pode definir o tipo de vista (mensal, semanal, diária), os nomes dos dias e meses, e restantes opções, facilmente identificadas na documentação.

3 Partial view é uma view que vai ser renderizada dentro de uma view pai. Estas são usualmente utilizadas para criar controlos costumizados pelo utilizador, tal como os .ascx no ASP.NET classic.

As propriedades mais importantes, e que vale a pena referir, são a *dayClick* e *eventClick*, dado que permitem realizar ações, após cliques no calendário ou em eventos.

Flot

Esta é mais uma biblioteca jQuery fácil de trabalhar.

Tal como na FullCalendar, na View basta colocar uma div onde irá ser construído o gráfico, via JavaScript. O método de construção no JavaScript é idêntico, ou seja, uma chamada ajax para não atrasar o carregamento da página e com os dados devolvidos pela Action. Aqui, a construção varia conforme o tipo de gráfico que se queira.

Como o construtor está adaptado para desenhar diferentes tipos de gráficos, os dados e a configuração do gráfico também são diferentes. Então, o construtor do gráfico recebe três argumentos: uma variável com a div, uma variável com os dados no formato correto perante o tipo de gráfico a desenhar e, por último, uma variável com as configurações do gráfico. É nesta última variável que se identifica o tipo de gráfico, alguns aspetos de design, p.e., se o gráfico é clicável ou não. Para esta propriedade funcionar, é necessário fazer o mapeamento de uma função ao gráfico para este tipo de ação.

4.1.9 *Permissões*

Devido às diferentes exigências de clientes e ao grande número de possíveis utilizadores e diferentes perfis, houve a necessidade de conseguir controlar o que é visível, editável, e que ações se podem realizar, tanto por perfil como por utilizador. Sendo assim, esta informação teria de ser guardada em base de dados e tratada aquando da execução da plataforma.

Existem três áreas distintas onde esta abordagem teve de ser realizada.

Permissões por área

Considera-se um Edit um conjunto Model-View-Controller que seja para apresentar e editar informação de uma entidade bem definida, p.e., um aluno ou uma turma.

Para tornar a apresentação mais apelativa, a informação deve ser dividida em grandes áreas (HorizontalTabs) e dentro de cada grande área, pequenas áreas (VerticalTabs). Mais à frente apresentar-se-á o resultado visual deste conceito.

Com a informação guardada na base de dados, estas tabs serão visíveis ou não e os campos de cada área serão editáveis ou não.

Para tal ser possível, o modelo de cada página irá estender o EditModel, modelo que irá guardar a informação das tabs que o utilizador tem permissões para visualizar e quais delas pode editar. Esta informação é guardada numa string JSON, que irá ser passada como parâmetro, já na View, a uma função JavaScript que irá tratar os dados, escondendo as tabs não definidas e colocando os campos não editáveis nas áreas que o utilizador apenas pode visualizar. A.2

Se o utilizador não tiver nenhuma área editável, o botão de Guardar será escondido.

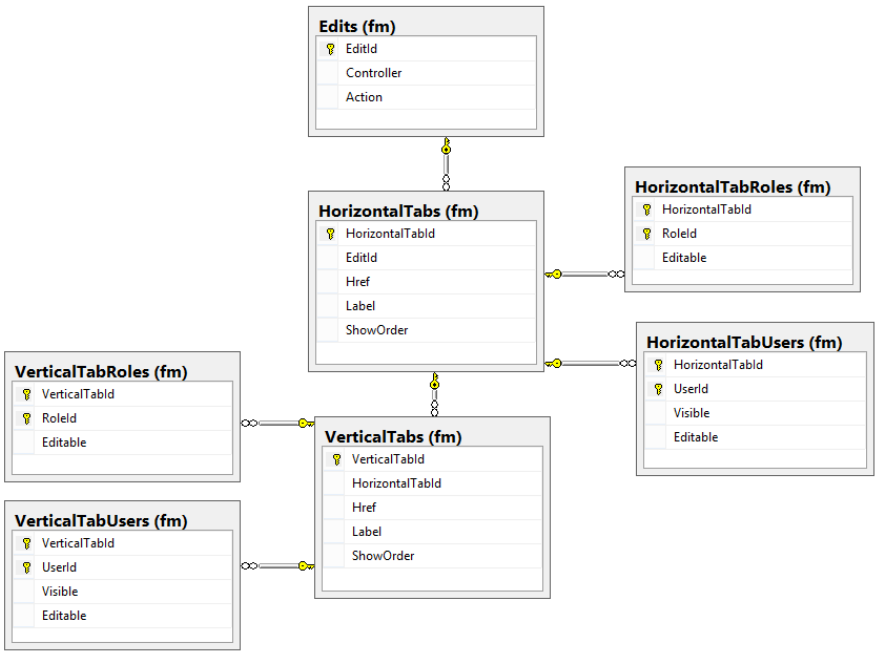


Figura 7.: Diagrama BD Horizontal e Vertical Tabs

Ações por edit

Num Edit, a ação normal é a de guardar as alterações dos campos feitas pelo utilizador. Esta, tal como referido acima, é acionável caso o utilizador tenha permissões para editar uma ou mais áreas. Para além desta ação, podem ser definidas outras ações que façam sentido haver no contexto do Edit. Assim, introduz-se o conceito de SmallAction.

Um Edit terá um conjunto de SmallActions acessíveis por um botão no canto superior deste. Estas são guardadas na base de dados, tal como indica o diagrama, contendo informação de quem as pode utilizar, por perfil e por utilizador.

Para tal ser apresentado em todas as páginas, criou-se um conjunto Model-View-Controller, que será invocado no Layout, mas que apenas é desenhado se o Edit, onde nos encontramos, contiver SmallActions. A Action chamada, tem de, através do ControllerContext, descobrir qual é o seu Edit pai para se colocar as ações corretas no Edit invocado.

```
string controllerName = this.ControllerContext.ParentActionViewContext.RouteData.
    Values["controller"].ToString();
string actionName = this.ControllerContext.ParentActionViewContext.RouteData.
    Values["action"].ToString();
object p = this.ControllerContext.ParentActionViewContext.ViewBag.
    SmallActionParameters;
```

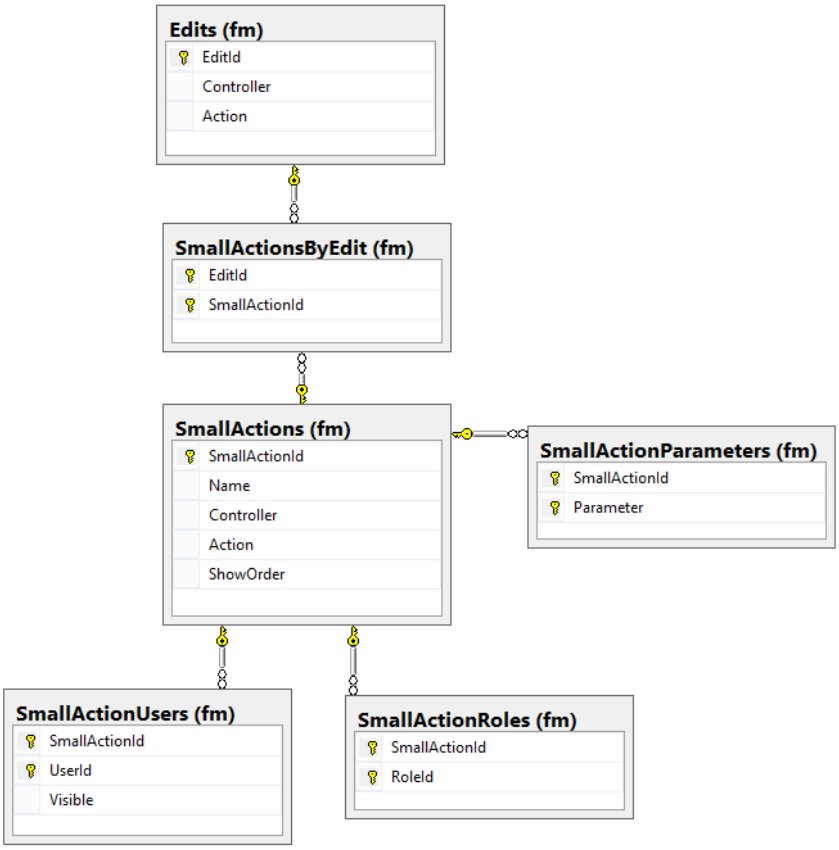


Figura 8.: Diagrama BD SmallActions

Estas ações, como seria expectável, necessitam de parâmetros de contexto. Estes parâmetros são colocados na ViewBag do Edit consoante são necessários.

```
object obj = new { classId = "classIdParam", studentId = "studentIdParam", userId
     = 307 };
ViewBag.SmallActionParameters = obj;
```

Com esta informação já é possível chamar o construtor do modelo, que tem como única propriedade uma lista de UrlAction. O método principal é apresentado como anexo A.3.

Como se pode verificar, é utilizado *reflection* para inferir as propriedades dos parâmetros de contexto e faz-se a correspondência com os parâmetros que cada SmallAction necessita (informação trazida da base de dados).

Para finalizar, cada UrlAction necessita de ter codificada a respetiva Action para realizar a ação pretendida.

Datatables

Já se falou nas DataTables e das suas possibilidades 4.1.7. Como este controlo é de bastante importância, houve necessidade de o tornar mais personalizável, distinguindo perfis e utilizadores. Assim, existe a possibilidade de esconder a tabela por perfil ou utilizador. Se esta for visível, certas colunas podem não ser visíveis, bem como as ações.

Estas opções foram colocadas no objeto estendido e as suas definições têm de ser guardadas na base de dados.

A visibilidade da tabela consiste apenas em verificar na base de dados se o utilizador está autorizado ou não a visualizá-la. Quanto às colunas, após verificar as definições para o utilizador em questão, é necessário alterar as suas definições geradas previamente, a partir das propriedades do objeto representado na tabela. Após adquirir da base de dados as ações que o utilizador está autorizado a realizar, é necessário fazer uma função JavaScript, que ficará encarregue de chamar a ação definida e enviar os IDs selecionados para, do lado do servidor, ser realizada a ação pretendida.

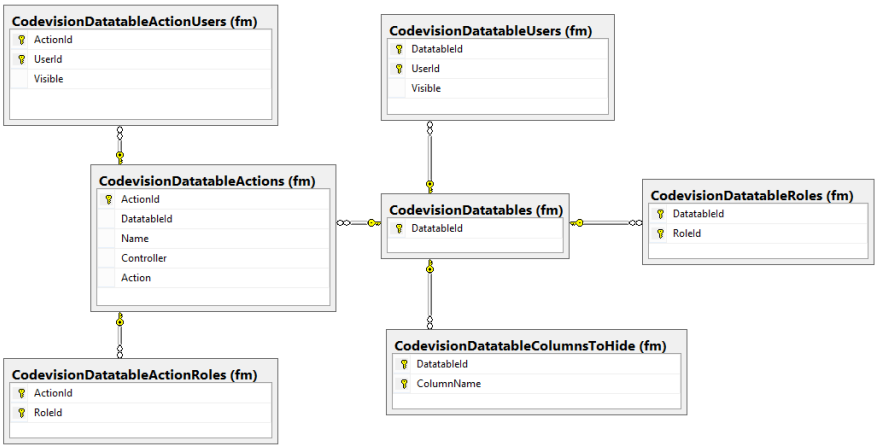


Figura 9.: Diagrama BD DataTables Permissions

4.2 WIN8 APP

A Microsoft fornece várias possibilidades para o desenvolvimento de aplicações Windows 8:

- JavaScript and HTML5
- C# and Extensible Application Markup Language (XAML)
- Microsoft Visual Basic and XAML
- Visual C++ component extensions (C++/CX) and XAML
- C++/CX and Microsoft DirectX

A escolhida foi a primeira, JavaScript and HTML5, por razões de conhecimento e de futura reutilização.

4.2.1 *Arquitetura*

A arquitetura JavaScript and HTML5, apresentada na Figura 10, tem um ponto de entrada na aplicação (default.html) e a partir daqui pode-se navegar para as páginas criadas. Cada página tem a sua pasta única e cada pasta tem três ficheiros: um ficheiro HTML, um ficheiro CSS e um ficheiro JavaScript.

Nos ficheiros HTML e CSS cria-se a interface do usuário (UI) da aplicação e o ficheiro JavaScript será uma espécie de *code-behind* do ficheiro HTML. É neste ficheiro que se fazem as funções necessárias para comunicar com o web service de modo a alimentar a página com os dados pretendidos.

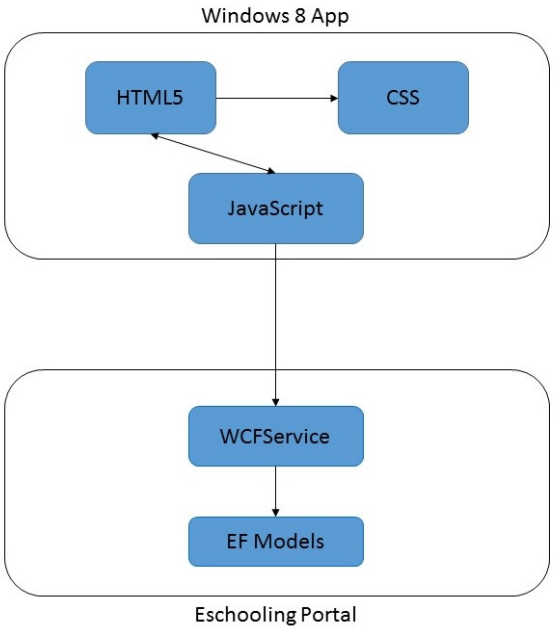


Figura 10.: Arquitetura Win8 app

4.2.2 *Navegação*

A navegação entre páginas pode ser feita de várias formas, sendo uma delas a tradicional: navegação utilizando a propriedade *href* do HTML, esquematizada na Figura 11. No entanto, esta não é a forma mais indicada de o fazer numa aplicação Windows 8, pois perde-se o estado da aplicação.

Utilizando a biblioteca *navigation.js* é também bastante fácil realizar navegação entre páginas e o estado não se perde. Ver esquema da Figura 12. Este estado é aquilo que se pode colocar na página de entrada da aplicação, default.html. Ou seja, CSS e JavaScript, comum a toda a aplicação, podem ser carregados aqui para não haver repetição em todas as páginas. A página para a qual se navega é carregada numa div, geralmente designada por *contenthost*, que se encontra na default.html.

Para se realizar a navegação basta utilizar um comando com a página pretendida como primeiro parâmetro e, facultativamente, um segundo parâmetro que pode ser um objeto JavaScript com o que se pretender.

```
WinJS.Navigation.navigate("/pages/dashboard/dashboard.html", data);
```

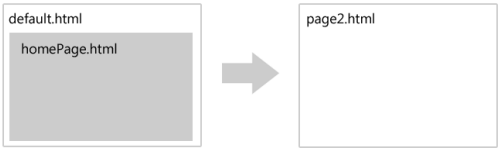


Figura 11.: Navegação tradicional

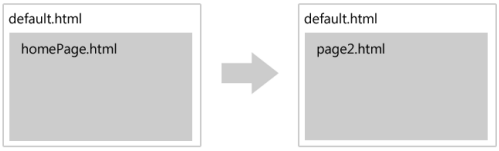


Figura 12.: Navegação com biblioteca navigation

4.2.3 *Web Service*

O web service para servir dados à aplicação Windows 8 foi feito em Windows Communication Foundation (WCF). Esta framework segue o design pattern Service-oriented architecture (SOA), ou seja, serve qualquer tipo de aplicação desde que os requisitos para estabelecer ligação sejam cumpridos.

Este serviço tem de estar sempre disponível devendo, então, estar hospedado no Internet Information Services (IIS) ou numa aplicação. Neste caso optou-se por utilizar como hospedeira uma aplicação, ESchooling Portal, que por sua vez está hospedada em IIS. Esta decisão teve como fator preponderante a ligação à sua camada de dados, ou seja, os dados que são necessários para fornecer à aplicação Windows 8 já estavam mapeados e facilmente acessíveis nesta aplicação.

Os métodos criados no Web Service são simples métodos de preenchimento de objetos e que são retornados. A única diferença é a assinatura do método, pois têm anotações para indicar que são métodos de um web service e devem respeitar um certo contrato e o formato da sua resposta. Neste caso optou-se por utilizar JSON, pois não são necessárias cuidados especiais na construção dos objetos e a sua reconstrução do lado da aplicação é fácil.

```
[OperationContract]
[WebGet(ResponseFormat = WebMessageFormat.Json)]
```

4.2.4 *Pedido e apresentação de dados*

Para melhor se perceber como funciona a comunicação entre a aplicação Windows 8 com o Web Service e como são tratados os dados de modo a estes ficarem apresentáveis ao utilizador, tome-se, como exemplo, a construção de uma lista de disciplinas, que compõe o plano curricular do aluno.

É feita uma chamada ajax ao web service A.4, invocando o método que devolve o plano curricular. Se o método *GetCurricularPlan* A.5 do lado do web service funcionar corretamente e os dados chegarem sem problemas à aplicação, tem de se gerar o html necessário para fazer a lista das disciplinas e colocá-las no elemento apropriado. Para além da questão do desenho da informação, também se deve tratar as ações pretendidas, quando se interage com estes elementos, daí a definição de uma função para cada disciplina, aquando do despoletamento da função onClick do elemento.

5

APLICAÇÕES

Neste capítulo serão apresentadas as aplicações que suportam o trabalho desenvolvido. Várias imagens servirão de apoio visual a conceitos referidos anteriormente, apresentando, assim, os controlos desenvolvidos, o aspeto geral das aplicações, em que tipo de janelas se aplicam as permissões configuráveis por perfil e utilizador e como se comporta a navegação.

5.1 FLATMOUNTAIN

O FlatMountain será uma alternativa mais leve para o professor e para o diretor de turma do que o E-Schooling. Todas as funcionalidades que estes podem realizar no E-Schooling podem também fazê-lo no FlatMountain, sendo esta uma aplicação mais focada às suas necessidades. O E-Schooling, com a sua vertente para perfis como secretaria, diretor pedagógico, financeiro, é uma aplicação mais focada a configurações base que os professores e diretores de turma não têm acesso. Sendo assim, o FlatMountain tem um design diferente em termos de organização de menus e novos controlos.

O *widget*[1] (Figura 13) de acesso rápido serve para, ainda na página principal, o utilizador ter acesso aos sumários das suas aulas, que estão a decorrer. Serão estes que irão ser mais acedidos, daí a sua colocação nesta página. Ainda nos *widgets* de acesso rápido, os diretores de turma, podem aceder às faltas e ocorrências dos alunos da sua direção de turma. Esta, também, é considerada informação importante e que necessita de destaque, pois os diretores de turma têm de estar sempre ao corrente desta informação, a fim de notificarem os encarregados de educação e tomarem as medidas necessárias.

Ainda na página principal, o utilizador pode consultar o seu calendário (Figura 14). Este tem toda a sua agenda, desde aulas, atividades, reuniões e conselhos de turma. Ainda neste controlo, o utilizador pode criar novas reuniões, convocando outros utilizadores da plataforma. Estes receberão uma mensagem com a notificação de nova reunião e esta aparecerá no seu calendário.

As tabelas (Figuras 15 e 16) são um controlo muito importante no E-Schooling, dada a quantidade de informação existente. Assim, este controlo tem de ter uma boa usabilidade e uma boa performance.

1 Widget é um pequeno controlo apresentado numa página que concentra alguma informação e possibilita acesso rápido a alguma funcionalidade.

Figura 13.: Widget no dashboard

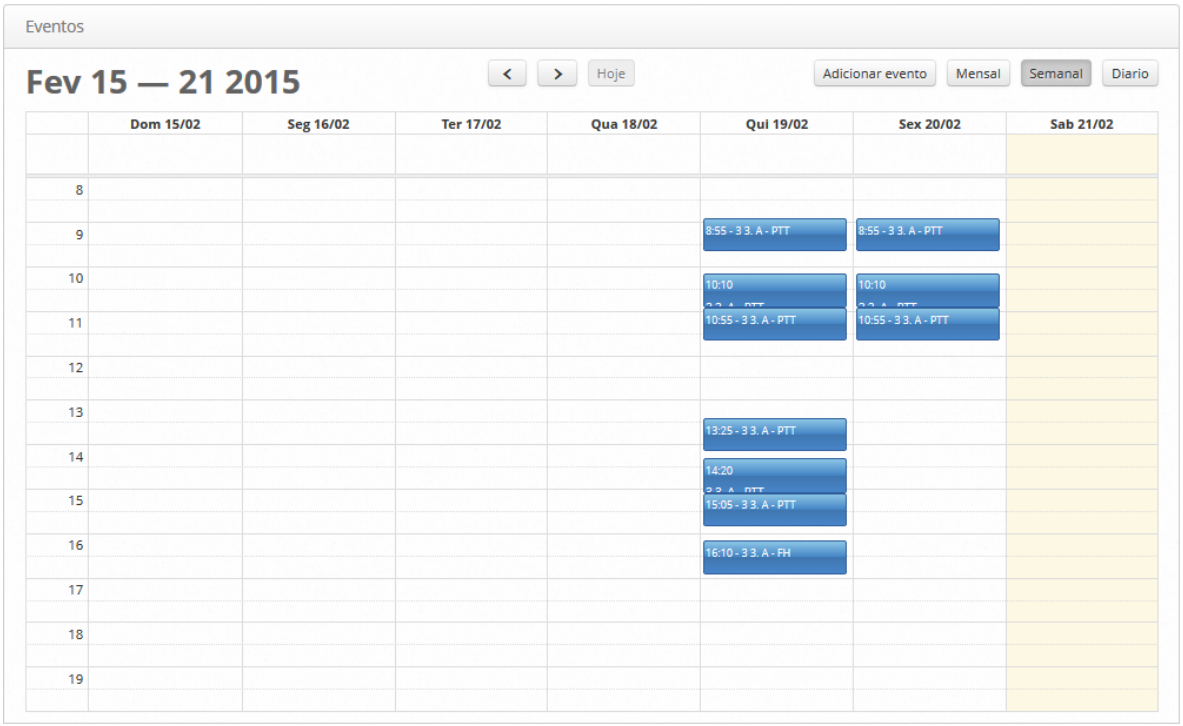


Figura 14.: Calendário de eventos

Tal como referido, devido à quantidade de informação existente, cada tabela pode ter imensos registos, o que torna a sua pesquisa fundamental (Tidwell, 2011). Assim, decidiu-se disponibilizar dois tipos de pesquisa: geral (Figura 15) e por colunas (Figura 16).

A pesquisa geral irá procurar em todas as colunas simultaneamente. Será utilizada quando o utilizador sabe que está a procurar algo unívoco e torna o design mais limpo. A pesquisa por colunas pode ser feita carregando no botão "Mostrar/Esconder Filtros". Esta fará a pesquisa apenas na coluna onde se introduz o pretendido e é cumulativa, ou seja, pode-se pesquisar em vários colunas simultaneamente, pois a tabela irá fazer uma filtragem dos dados, segundo os filtros introduzidos em cada coluna.

Todas as pesquisas são feitas de imediato à medida que o utilizador introduz o filtro. Estas são server-side, pois, apesar de os dispositivos pessoais estarem cada vez mais potentes, uma tabela com muitos registos prejudicaria a usabilidade ou, em casos extremos, provocaria a interrupção na aplicação.

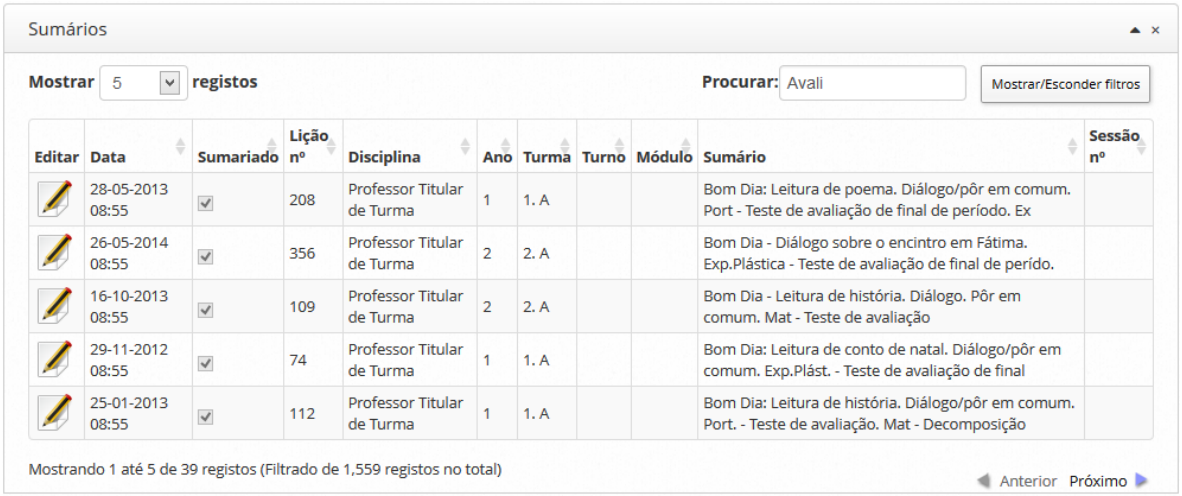


Figura 15.: Pesquisa geral nas tabelas

Na Figura 17 pode-se ver um simples Edit 4.1.9, talvez o mais usado da aplicação, que possibilita a um professor a escrita de um sumário, a marcação de faltas e ocorrências aos alunos. A marcação de faltas e ocorrências são assíncronas para o utilizador não ter de esperar pela resposta do servidor, quando pretende fazer vários registos, poupando assim tempo.

O gráfico de faltas da Figura 18 do docente permite uma análise geral das suas faltas (Tidwell, 2011). Para um maior detalhe, este pode carregar numa das barras, indicando-lhe informação individual de cada falta e possibilitando-lhe a sua justificação.

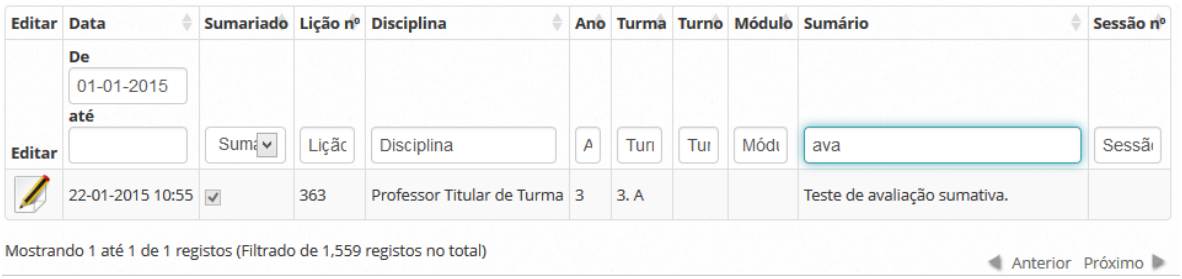


Figura 16.: Pesquisa por colunas nas tabelas

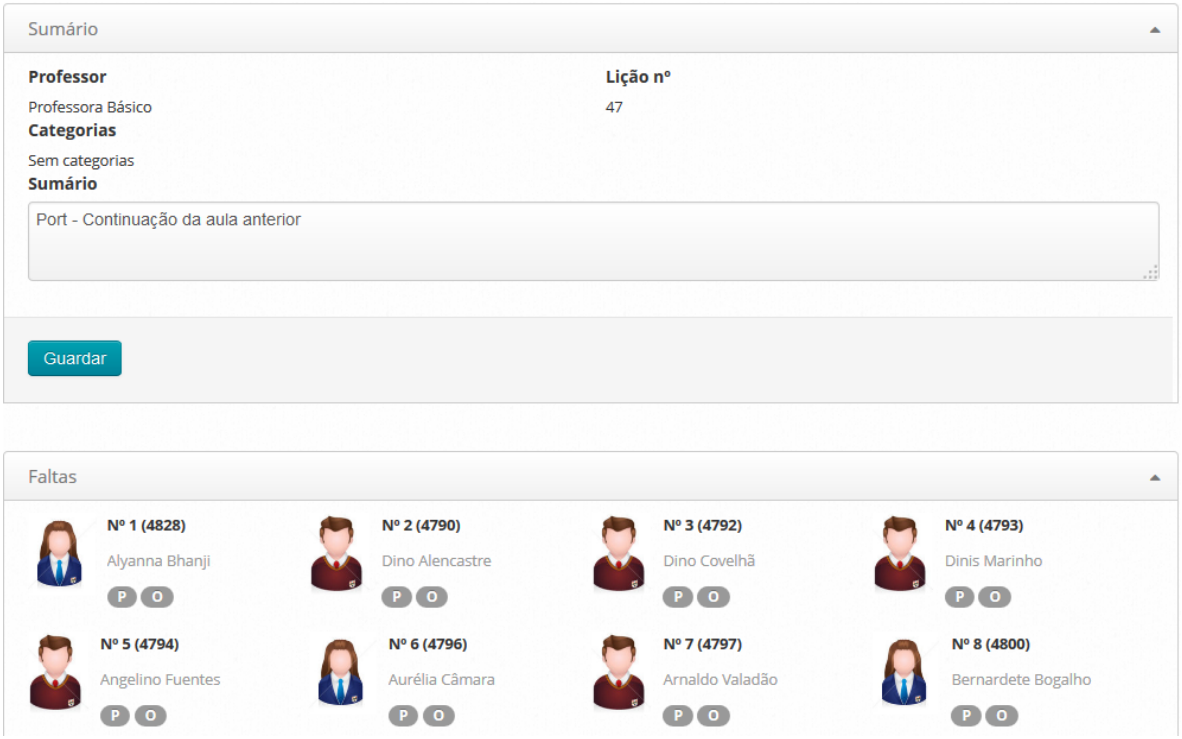


Figura 17.: Escrever sumário

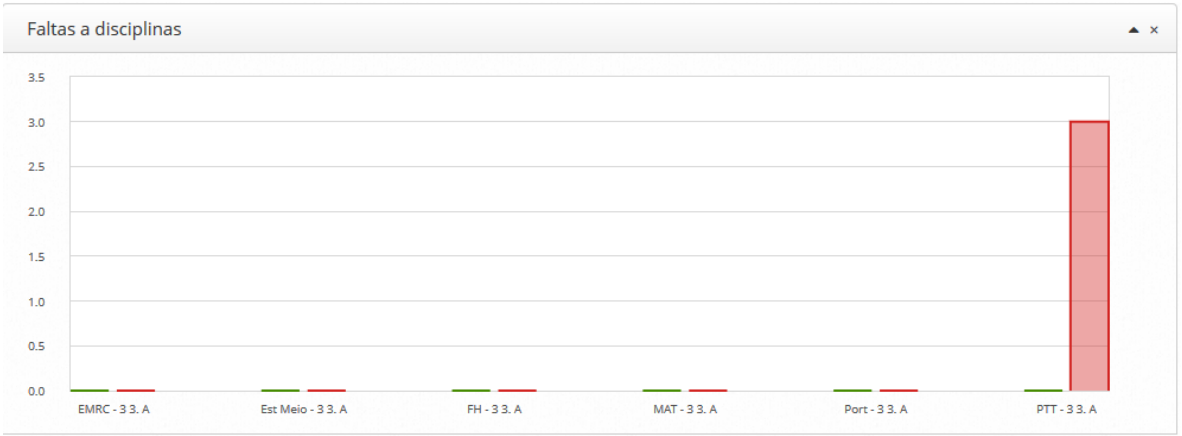


Figura 18.: Faltas do docente

Na Figura 19 pode-se ver um Edit mais complexo e já com lógica de permissões. Consegue-se identificar as tabs horizontais e, tal como referido anteriormente, as suas filhas, tabs verticais. É sobre estas tabs e, consequente área de edição, que é possível alterar as permissões ao perfil e ao utilizador de visionamento e edição.

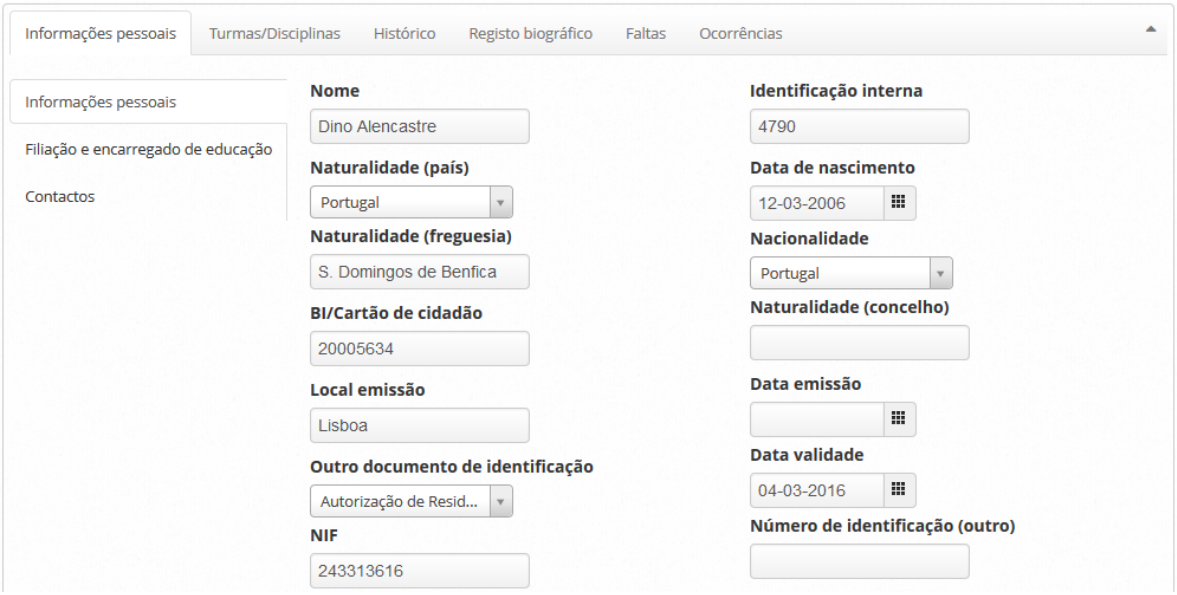


Figura 19.: Edit - geral

Dois controlos importantes, e no futuro de fácil implementação, são a TextBox AutoComplete 20 e a Lookup 21. Estes controlos também são muito importantes para o utilizador, pois conseguem pesquisar e filtrar um elemento de um grande universo com relativa facilidade e boa performance (Tidwell, 2011). Também aqui a computação é feita do lado do servidor.

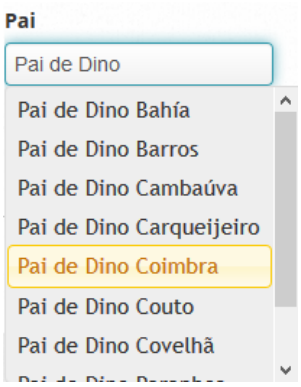


Figura 20.: Controlo TextBox AutoComplete

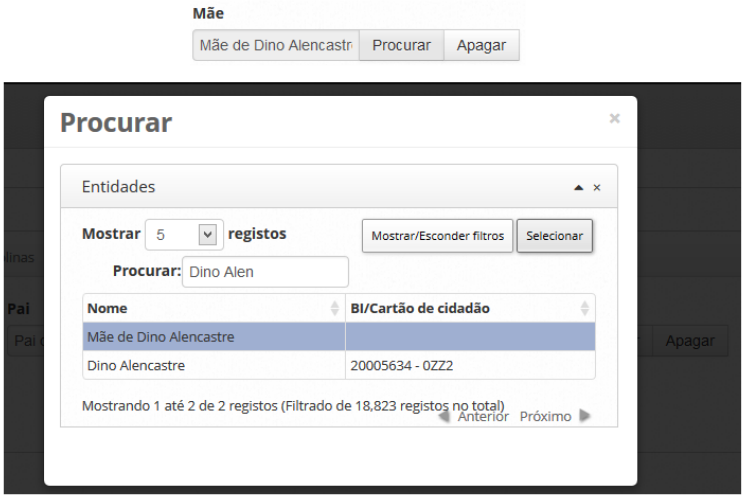


Figura 21.: Controlo Lookup

Uma das funcionalidades mais interessantes desta aplicação é o chat. Este pode ser apenas com um utilizador, ver Figura 22, ou com um grupo de utilizadores, ver Figura 23. Um utilizador pode criar

um grupo e adicionar os utilizadores que quiser, carregando com o botão direito do rato (ou premindo durante algum tempo em caso de *touch*) em cima do utilizador pretendido e escolher qual o grupo, dos quais administra, onde pretende adicionar o utilizador selecionado. Este consegue ver os grupos dos quais é administrador (em cima) e àqueles que pertence mas não é administrador (em baixo).

O chat, apresentado na Figura 24, é *real-time*, ou seja, o utilizador recebe a mensagem mesmo não fazendo *refresh* à página. Caso o chat com o utilizador que lhe enviou a mensagem não esteja aberto, este recebe uma notificação no topo da página.

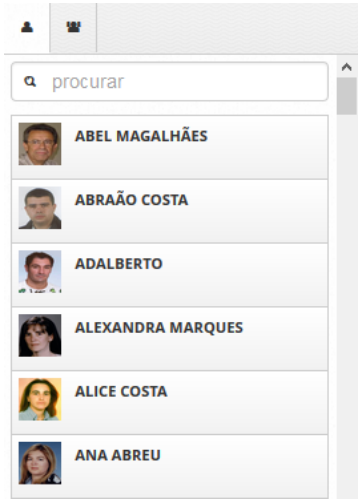


Figura 22.: Chat Users

Figura 23.: Chat Groups

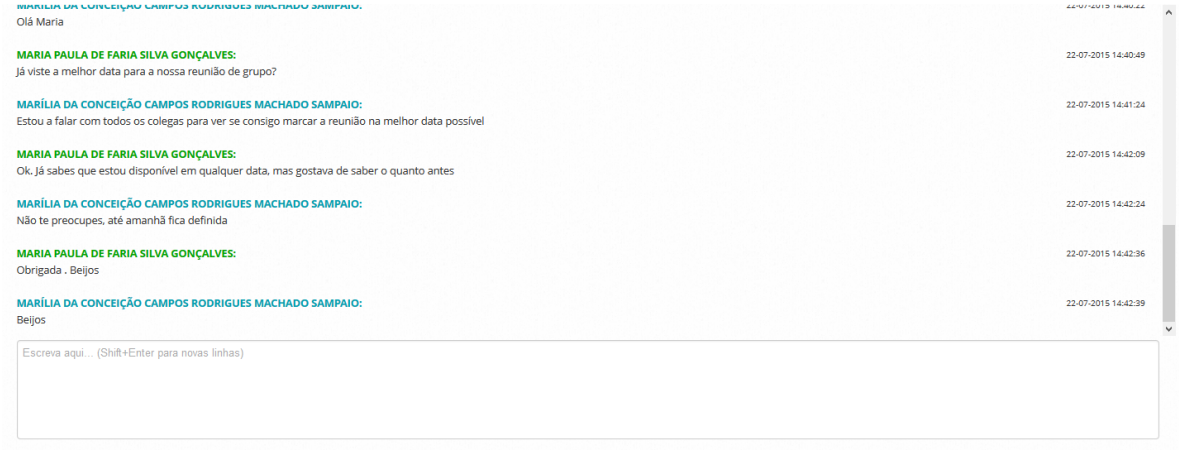


Figura 24.: Chat

5.2 WIN8 APP

Esta é uma aplicação que pretende tirar o máximo partido dos Microsoft Surfaces a que os alunos tiveram fácil acesso. Pensada como um complemento ao caderno, os alunos podem consultar o seu horário, tirar notas, partilhar documentos, descarregar documentos disponibilizados pelo professor, trocar mensagens entre si, entre outras funcionalidades.

Sendo uma aplicação Windows 8, o seu scroll é horizontal tal como se pode verificar nas imagens ilustrativas do dashboard 25 e 26. No lado esquerdo do dashboard (Figura 25), pode-se ver um menu principal, onde o aluno pode aceder a todas as áreas da aplicação, a sua agenda diária e as suas notas mais recentes. No lado direito (Figura 26), os eventos futuros e as últimas mensagens trocadas.

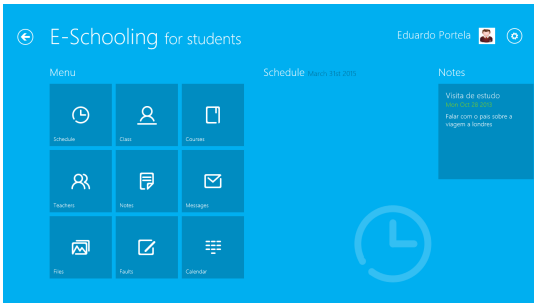


Figura 25.: Dashboard Left

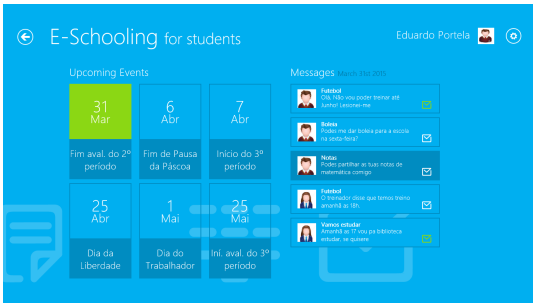


Figura 26.: Dashboard Right

No menu das disciplinas, apresentado na Figura 27, o aluno tem acesso ao seu plano curricular, podendo ter acesso a informação individual de cada disciplina, tal como: quem é o professor que leciona essa disciplina, as suas faltas, as suas classificações e acesso às notas, associadas àquela disciplina.

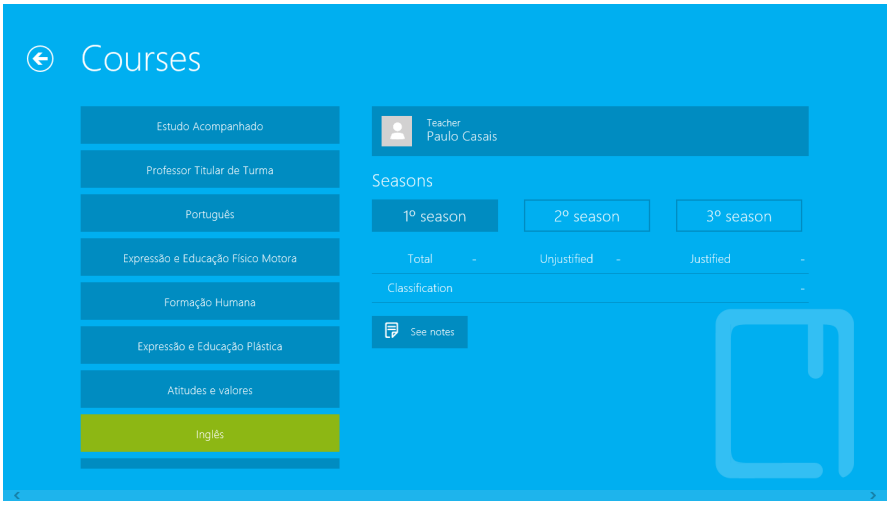


Figura 27.: Courses

No menu de notas, que se pode ver na Figura 28, o aluno acede a todas as suas notas. Estas podem ser filtradas pela associação à disciplina, por data ou por texto no seu conteúdo. A cada nota podem estar associados ficheiros, tais como .pdf, .xls ou .doc, que podem ser descarregados. Estes ficheiros podem ser carregados pelo professor ou pelo aluno.

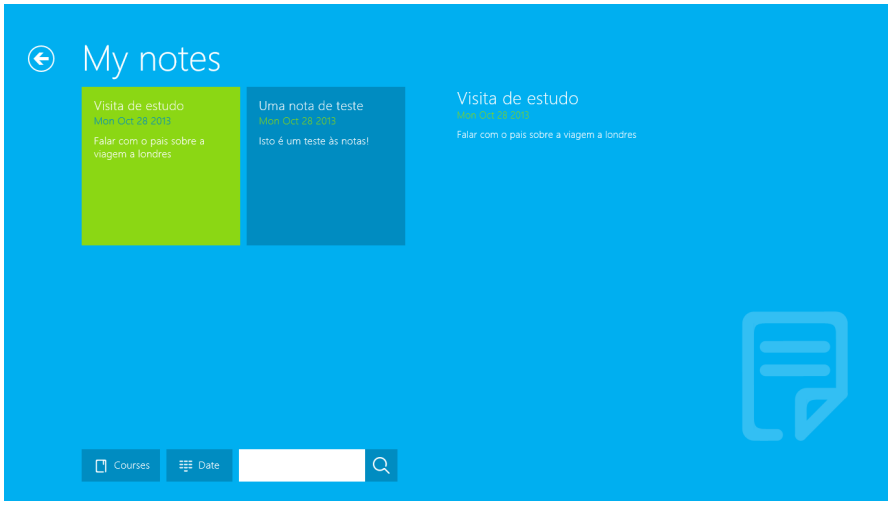


Figura 28.: Notes

No menu das mensagens, apresentado na Figura 29, o aluno verifica a troca de mensagens que fez com os colegas.

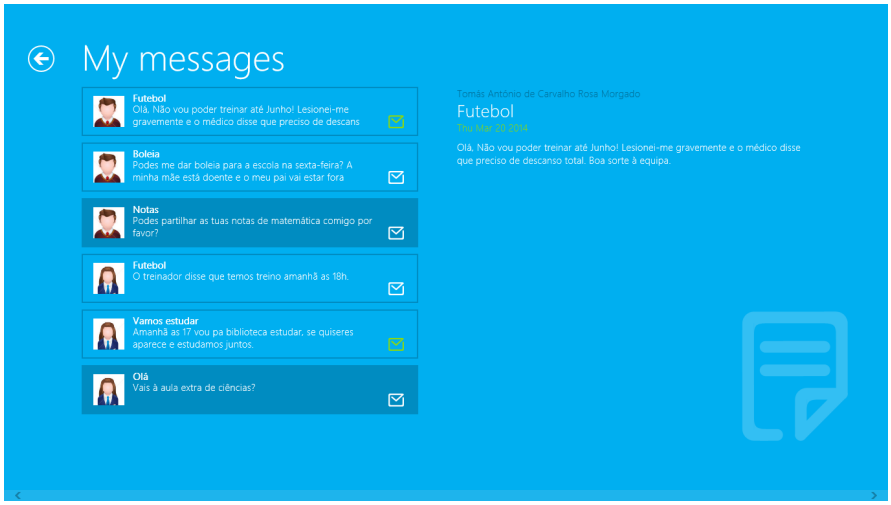


Figura 29.: Messages

6

CONCLUSÕES E TRABALHO FUTURO

6.1 CONCLUSÕES

Os objetivos desta dissertação - estudar a melhor arquitetura para dois tipos de aplicações distintos; desenvolver um middleware estendendo funcionalidades das frameworks de desenvolvimento e de bibliotecas; aplicar este estudo e desenvolvimento à concretização de duas aplicações - foram atingidos. Neste momento, as duas aplicações estão numa fase de acréscimo de funcionalidades ao utilizador, provando assim a solidez das bases criadas.

Com o aprimorar dos controlos e o desenvolvimento dos vários helpers, o middleware do Flat-Mountain oferece aos programadores grande facilidade para desenvolverem novas funcionalidades. Esta base é importante para uma maximização do rendimento de cada programador e deve ser algo que todos os programadores nesta framework devem explorar. Sendo uma camada tão importante, tem de ser mantida num processo de evolução constante.

Quando se desenvolve uma aplicação web, a questão de que lado é feita a computação é algo muito importante. Neste sentido, uma distribuição da carga é sempre o desejável. Aqui, evitou-se o sobrecarregamento de elementos do lado do cliente, que provocaria uma má experiência de utilização, fazendo pedidos assíncronos ao servidor e construindo do lado do cliente apenas os elementos necessários à apresentação. Pequenas interações e verificações, são feitas do lado do cliente, evitando uma sobrecarga de pedidos ao servidor.

Algo que deve ser pensado, ainda antes de começar a desenvolver este tipo de aplicações, é a possibilidade que estas têm de atravessar fronteiras. Sendo assim, a preparação da aplicação com dicionários deve ser feita de raiz, tornando o processo de disponibilização de uma nova língua algo tão simples como traduzir todas as entradas no dicionário. Assim, este trabalho pode ser delegado a alguém que não um programador, podendo este focar-se em algo mais importante. Caso o desenvolvimento comece por ser feito estaticamente apenas numa língua e mais tarde se deseje internacionalizar o produto, o processo tornar-se-ia demorado e poderia levar a falhas de tradução em alguns elementos, bem como a problemas relacionados com datas e moedas.

Na fase da arquitetura da aplicação, decidiu-se por um acesso direto à base de dados, em detrimento da utilização de um webservice. Contudo, esta decisão ainda está por provar como tendo sido a mais acertada.

Relativamente à aplicação Windows 8, a Microsoft tornou o processo de aprendizagem quase nulo para quem está habituado a desenvolver para ambiente web. Esta abordagem é surpreendente e retira um *overhead* enorme no tempo que é necessário para desenvolver uma aplicação nativa, caso os programadores, que uma empresa tenha disponíveis, não saibam a linguagem necessária. Neste sentido, a Microsoft fez um bom trabalho e facilitou todo o processo para a publicação de aplicações na sua *store*, faltando agora ganhar *market share* para que se verifique um aumento de aplicações e se torne rentável às empresas o investimento necessário em recursos para lançar este tipo de aplicações

6.2 TRABALHO FUTURO

Tendo como objetivo lançar o FlatMountain para total utilização por parte dos clientes até Setembro, as funcionalidades em falta, e que estão presentes no E-Shooling para os perfis de docente e diretor de turma, devem estar disponíveis até ao prazo estabelecido.

Algo que deve ser experimentado é o alojamento da aplicação em Azure e uniformizar o seu acesso a apenas uma aplicação, ou seja, em vez de existir uma aplicação para cada cliente, existir uma disponibilizada pela Codevision. Esta abordagem implicaria algumas alterações, começando pelo principal problema encontrado: a diferenciação dos utilizadores de clientes diferentes.

Com o evoluir das frameworks por parte da Microsoft, e tentando acompanhar esta evolução, a migração da aplicação de MVC4 para MVC5 (ou até MVC6 que está em fase beta) deve ser tida em conta. Uma das principais alterações, e que tem de ser estudada, é a funcionalidade dos Tag Helpers, em alternativa aos MVC Helpers. O argumento utilizado pela Microsoft para esta alteração é a maior semelhança com HTML puro, facilitando assim o trabalho de programadores front-end.

Algo que seria interessante testar com a aplicação Windows 8, aproveitando a sua arquitetura que utiliza apenas linguagens front-end, que possibilita tal teste, é a utilização de uma ferramenta como o PhoneGap para converter a aplicação em aplicações nativas iOS e Android, a fim de testar a sua usabilidade e as diferenças existentes para aplicações puramente nativas.

A aplicação windows 8 ainda deve evoluir no sentido da integração com o E-Schooling e o E-Schooling Portal, aumentando as suas funcionalidades e tornando-a algo imprescindível ao aluno na sua interação com a escola.

BIBLIOGRAFIA

J. Galloway, P. Haack, B. Wilson, and K. S. Allen. *Professional ASP.NET MVC4*. Wiley, 2012.

Gartner. Gartner says worldwide tablet sales grew 68 percent in 2013, Março 2014. URL http://www.gartner.com/newsroom/id/2674215.

Perry Marchant. Performance and the entity framework, Agosto 2009. URL http://www.codeproject.com/Articles/38922/Performance-and-the-Entity-Framework.

E. Marcotte. Responsive web design, 2010. URL http://alistapart.com/article/responsive-web-design.

Microsoft ASP.NET Team. Asp.net mvc overview, Janeiro 2009. URL http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview.

J. Tidwell. *Designing Interfaces*. O'Reilly, 2nd edition, 2011.

Jeremy Weinstein. Make the mobile web faster, Agosto 2012. URL https://developers.google.com/speed/articles/mobile.

Parte III

APÊNDICES

A

EXEMPLOS DE CÓDIGO

```
private static string ActionToLookupSelection(string dataTableId, string
    fieldName, string selectedTextPropertyName, IEnumerable<ColDef> columns)
{
  int selectedTextPropertyIndex = -1;

  // need the index of the selected registry to put it right away in the textBox
     without a call to the server
  for (int i = 0; i < columns.Count() && selectedTextPropertyIndex == -1; i++)
  {
    ColDef cd = columns.ElementAt(i);
    if (cd.Name == selectedTextPropertyName)
      selectedTextPropertyIndex = i;
  }

  var action = new { sExtends = "text", sButtonText = new JRaw("Resources.Select"
     ), fnClick = new JRaw(CreateJavaScriptFunctionToLookupSelection(dataTableId,
      fieldName, selectedTextPropertyIndex)) };
  return JsonConvert.SerializeObject(action, Formatting.Indented);
}

private static string CreateJavaScriptFunctionToLookupSelection(string
    dataTableId, string fieldName, int selectedTextPropertyIndex)
{
  string f = "function (nButton, oConfig, oFlash) {";
  f += "var oTT = TableTools.fnGetInstance('" + dataTableId + "');";
  f += "var aData = oTT.fnGetSelectedData();";
  f += "var selectedRow = aData[0];"; // single select, only got 1 Id
  f += "var selectedRowId = selectedRow[0];";
  f += "var selectedText = selectedRow[" + selectedTextPropertyIndex + "];";
  f += "$('#" + fieldName + "').val(selectedRowId);";
  f += "$('#tb" + fieldName + "').attr(\"value\", selectedText);";
  f += "$('#modal" + fieldName + "').find(\"button\", \".close\")[0].click()";
  f += "}";

  return f;
}
```

Exemplo de código A.1: DataTableLookupDefinition

```
function turnAllFieldsReadOnly(divId) {
    // tratar inputs - textBoxes, checkBoxes, datePickers, timePickers
    $(divId).find('input').each(function () {
        this.readOnly = true;
        if (this.type == "checkbox") {
            this.setAttribute("onclick", "return false;");
        }
    });
```

```
    // tratar selects - dropDownLists
    $(divId).find('select').each(function () {
        var input = document.createElement("input");
        input.setAttribute("type", "hidden");
        input.setAttribute("name", this.name);
        input.setAttribute("value", this.options[this.selectedIndex].value);
        $(divId)[0].appendChild(input);

        this.disabled = true;
    });

    // tratar textAreas
    $(divId).find('textarea').each(function () {
        this.readOnly = true;
    });
}

function applyAreasPermissions(tabs) {
    var isActiveTab = false;
    var countVisible = 0;
    var countEditable = 0;

    $(".nav-tabs").children().each(function () {
        var a = $(this).children("a")[0];
        var tabId = a.getAttribute("href").replace("#", "");
        var tabDef = tabs[tabId];
        var divId = "#" + tabId;

    // if previous active, the next one becomes the active one
        if (isActiveTab) {
            this.className = "active";
            $(divId)[0].className = $(divId)[0].className + ' active in';
            isActiveTab = false;
        }

        // if tab definition does not exist, then we do not see it. Hide the tab
            and the div
        if (tabDef == null) {
            this.style.display = "none";
            $(divId)[0].style.visibility = "hidden";

            // Check if actual tab is the active one
            // if so, it becomes non active, the div as well and save this
                information to activate the next one
            if (this.className == "active") {
                isActiveTab = true;
```

```
                this.className = "";
                $(divId)[0].className = "tab-pane fade";
            }
        }
        else {
            countVisible++;

            // if in tab definition, editable = false, all the fields become
                readOnly. Not disable because they must be POSTed
            if (tabDef.Editable == false)
                turnAllFieldsReadOnly(divId);
            else
                countEditable++;
        }
    });

    if (countVisible == 0 || countEditable == 0) {
        // Save button not visible
        $(".form-actions").children('button')[0].disabled = true;
    }
}
```

Exemplo de código A.2: ApplyAreasPermissions JavaScript functions

```
private List<UrlAction> BuildSmallActions(string controller, string action,
    object contextParams, int userId)
{
  IList<PropertyInfo> props = null;
  if (contextParams != null)
  {
    Type myType = contextParams.GetType();
    props = new List<PropertyInfo>(myType.GetProperties());
  }

  BLLModelSystem bll = new BLLModelSystem();
  IEnumerable<SmallAction> smallActions = bll.
      GetSmallActionsByControllerActionAndUser(controller, action, userId);
  List<UrlAction> smallActionsRes = new List<UrlAction>();

  foreach (SmallAction sa in smallActions)
  {
    List<SmallActionParameter> saps = sa.SmallActionParameters.ToList();
    RouteValueDictionary rvc = new RouteValueDictionary();

    foreach (PropertyInfo prop in props)
    {
      if (saps.Select(x => x.Parameter).Contains(prop.Name))
```

```
      {
        rvc.Add(prop.Name, prop.GetValue(contextParams, null));
      }
    }

    smallActionsRes.Add(new UrlAction()
    {
      Name = sa.Name,
      ControllerName = sa.Controller,
      ActionName = sa.Action,
      RouteValues = rvc,
      ShowOrder = sa.ShowOrder
    });
  }

  return smallActionsRes;
}
```

Exemplo de código A.3: BuildSmallActionsMethod

```
function getCurricularPlan(studentid) {
    var url = getAppUrl();
    $.ajax({
        url: url + '/WebServices/WCFService.svc/GetCurricularPlan',
        data: { studentId: studentid },
        cache: false,
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        responseType: "json",
        success: function (result) {
            if (result) {
                var html = "<ul class='courses-list'>";
                var length = result.length;
                for(var i = 0; i < length; i++){
                    html += "<li><a class='clickableCourse'> <input class=\"
                        courseIdHidden\" type='hidden' value='" + result[i].
                        CourseId + "'/> <input class=\"stIdHidden\" type='hidden'
                        value='" + studentid + "'/> <span class='desc'>" + result[
                        i].CourseName + "</span></a></li>";
                }
                html += "</ul>";
                $("#courses").html(html);

                var cc = $(".clickableCourse");
                $(cc).each(function (i, elem) {
                    if (i == 0) {
                        elem.onclick = function () { showCourseInfo(this); }
```

```
                        showCourseInfo(elem);
                    }
                    else {
                        elem.onclick = function () { showCourseInfo(this); }
                    }
                });

            }
            $('#divLoader').hide();
        },
        error: function () {
            $('#divLoader').html('<label>Error retrieving data</label>');
        }
    });
}
```

Exemplo de código A.4: getCurricularPlan JavaScript function

```
[OperationContract]
[WebGet(ResponseFormat = WebMessageFormat.Json)]
public List<StudentCourseCustom> GetCurricularPlan(string studentId)
{
  Guid stId = new Guid(studentId);
  List<StudentCourseCustom> scs = new List<StudentCourseCustom>();
  Eschooling dbe = new Eschooling();
  Guid syId = dbe.ScholarYears.Where(x => x.Active.Value).SingleOrDefault().
      ScholarYearId;

  List<StudentCourses> studentCourses = dbe.StudentCourses.Where(x => x.StudentId
       == stId).ToList();
  foreach (StudentCourses sc in studentCourses)
  {
    Courses c = dbe.Courses.Where(x => x.CourseId == sc.CourseId).SingleOrDefault
        ();
    Classes cl = dbe.Classes.Where(x => x.ClassId == c.ClassId).SingleOrDefault()
        ;
    if (cl.ScholarYearId == syId)
    {
      scs.Add(new StudentCourseCustom
      {
        CourseId = c.CourseId,
        CourseName = c.Name
      });
    }
  }

  return scs;
```

```
}
```

Exemplo de código A.5: GetCurricularPlan Web Service method