Mestrado em Engenharia Informática e Sistemas

Ramo de Desenvolvimento de Software

Evolução tecnológica de uma plataforma ASP.NET Web Forms para ASP.NET MVC

Relatório de Projeto

Dezembro 2014

a21130159 Tiago Filipe Marques Baía

Orientador ISEC: **Prof. Doutor João Cunha**

Agradecimentos

A todas as pessoas e instituições que contribuíram de alguma forma para a realização deste trabalho, os meus sinceros agradecimentos.

Ao Professor Doutor João Cunha agradeço pela sua orientação, com o profissionalismo e competência a que sempre me habituou nas várias vezes que nos cruzamos no meu percurso académico.

Á Critical Software agradeço o facto de me terem facultado o código fonte, documentação e infraestrutura necessárias para a realização deste trabalho. Um especial agradecimento aos meus colegas e amigos, Eng. Jorge Vieira, Eng. Luís Joaquim e Eng. Paulo Carreiro pela disponibilidade e colaboração demonstradas neste percurso.

Por fim, mas não menos importante, o constante suporte da minha família. Em particular da minha esposa Patrícia Jorge, que sempre me apoiou na minha necessidade constante de superar novos desafios académicos e profissionais.

Obrigado!

Resumo

As tecnologias estão hoje fortemente presentes no mundo que nos rodeia, e evoluem a uma velocidade impressionante. Um produto que hoje é o pico da inovação rapidamente cai em desuso e se torna obsoleto. O mesmo acontece com os processos de fabrico e arquitetura desses produtos.

O EMS é um produto da Critical Software vocacionado para a gestão de plantas de produção energética, que está já fortemente implantado na indústria de produção eólica e solar. Os grandes pontos fortes deste produto são a capacidade de monitorização e gestão centralizada de plantas heterogéneas dispersas geograficamente.

Este produto assenta sobre a plataforma ASP.NET Web Forms, da Microsoft, plataforma que, no momento de arranque do projeto que deu origem a este software, era a única abordagem disponível na *framework* Microsoft .NET para desenvolvimento Web. No entanto, desde então a Microsoft lançou a *framework* ASP.NET MVC, uma *framework* que possibilita a utilização do padrão de arquitetura MVC que, teoricamente, apresenta um conjunto de vantagens no desenho de interfaces.

A principal motivação deste trabalho foi a análise detalhada desta nova plataforma, com o intuito de perceber potenciais mais-valias que traria ao produto, seguida do estudo da capacidade de adaptar o EMS a esta nova arquitetura. É importante para o EMS perceber se de facto está a utilizar a melhor plataforma disponível atualmente.

Uma vez identificadas algumas potenciais melhorias na abordagem MVC, foi desenhada uma nova arquitetura com base no novo padrão. Para além da nova arquitetura de referência, foi também pensado e exercitado um processo de adequação a esta nova abordagem, capaz de ser introduzido no ciclo de vida do produto, com impactos minimizados na execução do *roadmap* existentes e nas atividades recorrentes da equipa de desenvolvimento.

Como resultado do trabalho realizado verificou-se que de facto existem potenciais mais-valias na utilização do padrão de arquitetura MVC no produto EMS. Foi desenhada uma nova arquitetura e encontrada uma solução que torna possível a coexistência das *frameworks* ASP.NET Web Forms e ASP.NET MVC na mesma solução, tornando a transição de tecnologias gradual.

No entanto a migração dos módulos de software existentes será sempre um processo com bastante impacto no projeto. É um processo demorado, e com alguns desafios técnicos complicados, como por exemplo a adequação da nova solução às bibliotecas de controlos fornecidos por terceiros.

Palavras-chave: software, arquitetura de software, MVC, ASP.NET, ASP.NET Web Forms, ASP.NET MVC, Scrum.

Abstract

Technology is present everywhere nowadays, and evolve at an impressive rhythm. A product that is today the innovation peak can quickly became obsolete. The same happens with manufacturing processes and architecture.

EMS is a product developed by Critical Software directed to energy production plants, strongly settled in the wind and solar industry. The strengths of this product are the capability of centralized monitoring and management of heterogeneous plants scattered geographically.

This product is based on Microsoft ASP.NET Web Forms platform. When the project of this product started this was the only available approach for Web development from Microsoft .NET Framework. However, since then, Microsoft launched ASP.NET MVC, a framework that allows developers to use the architecture pattern MVC which, theoretically, has advantages as interface development solution.

The main motivation for this work was the detailed analysis of this new platform, with the intention of acknowledge potential improvements for the product, followed by the study of the capacity to adapt EMS to this new architecture. It's important for EMS to understand if it's using the better available platform.

When identified potential improvements caused by the MVC approach, a new architecture was drawn based on this new pattern. Besides this new architecture, a new process for accommodation of the project to this new approach was designed and exercised, with the objective of being introduced in the product development life cycle, with minimal impact in the execution of the existing roadmap and the development and maintenance activities.

As result of this work some potential improvements in the MVC architecture pattern were identified. A new architecture was designed and a solution for coexistence of both ASP.NET Web Forms and ASP.NET MVC platforms in the same solution that makes the technology transition potentially less abrupt and intrusive.

However the migration of the existing modules will always be a process with great impact in the project. It's a long process, with some complicated technical challenges, like for example the adaptation in the new solution of some control libraries developed by external companies.

Keywords: software, software architecture, MVC, ASP.NET, ASP.NET Web Forms, ASP.NET MVC, Scrum.

Índice

Índice de Figuras

1 Introdução

As tecnologias estão hoje cada vez mais presentes em tudo o que nos rodeia, e evoluem de uma forma impressionante. Os produtos rapidamente se tornam obsoletos, e como tal é necessário um esforço de renovação e adequação constante às inovações tecnológicas.

Muitas das melhorias tecnológicas dos produtos existentes no meio que nos rodeia são conseguidas por inovações ao nível do software. As tecnologias de suporte ao software, nomeadamente as plataformas de desenvolvimento, estão constantemente a receber atualizações e evoluções que tornam isso possível.

O EMS (Energy Management System) é um produto de software desenvolvido pela Critical Software, uma das empresas de desenvolvimento mais conceituadas do país, fundada em 1998, e gere plantas de produção energética renovável. Esta solução está neste momento em produção em vários clientes, ajudando em todos o processo de gestão das plantas, que no total são já mais de 250, em 17 países, gerindo mais de 1 GW de energia.

O desenvolvimento deste produto teve início já há alguns anos, tendo com base um projeto de software feito à medida já existente. O EMS é no entanto um produto em evolução contínua, não só sob o ponto de vista da implementação de novas funcionalidades e correção de defeitos, mas também e em grande parte em termos de otimização e melhoria das funcionalidades nucleares, que já oferece.

Nesta fase, em que se perspetiva um longo período de investimento na evolução do produto, é interessante fazer um estudo da arquitetura do produto, para por um lado poder colmatar algumas falhas que possam ter sido herdadas do software original, e por outro poder evoluir a arquitetura para um padrão mais atual e mais adequado ao desenvolvimento de produto.

O EMS é um sistema complexo, e neste estudo o foco será a componente Portal, uma aplicação web desenvolvida na tecnologia Microsoft ASP.NET 4.0. Esta tecnologia é bastante popular hoje em dia, mas a Microsoft cada vez mais direciona o foco das suas plataformas de desenvolvimento para o padrão MVC, desde que lançou o ASP.NET MVC.

O principal objetivo deste estudo será uma análise comparativa entre a tecnologia atual do produto, o ASP.NET 4.0, e uma tecnologia mais atual lançada pela Microsoft para desenvolvimento Web, o ASP.NET MVC 4. Interessa encontrar uma abordagem possível para uma eventual evolução tecnológica, perceber os principais desafios que essa evolução representa e quais as mais-valias que daí poderão ser retiradas.

1.1 O produto EMS

Com o sector das energias renováveis cada vez mais preponderante na produção energética a nível mundial, as empresas produtoras de energia enfrentam desafios aliciantes, como a gestão de grandes e variadas plantas de produção energética, por vezes geograficamente dispersas, e com grande heterogeneidade ao nível dos equipamentos e respetiva configuração e manutenção em cada uma delas. Neste cenário, a ineficiência na deteção e correção de problemas ou a incorreta manutenção dos equipamentos irá originar um acréscimo nos custos, e uma menos eficiente produção energética.

A dispersão geográfica das plantas de produção energética tende a aumentar os custos de operação e manutenção, sendo necessário replicar ou deslocar os recursos, o que se torna dispendioso e ineficiente tendo em vista a capacidade de detetar a resolver problemas em tempo útil. Também a grande heterogeneidade que por vezes existe ao nível dos equipamentos de produção energética torna a gestão e manutenção mais demorada, requer mais pessoal e *know how*, e com tudo isto obviamente, torna-se mais dispendiosa e ineficiente.

É neste contexto que surge o produto EMS, um sistema de software que fornece funcionalidades ao nível da operação e manutenção de plantas energéticas.

O EMS oferece uma supervisão centralizada do funcionamento de plantas energéticas, completamente independente da localização geográfica e das especificidades técnicas dos equipamentos, sendo possível ter uma visão geral, e em tempo real, da energia a ser produzida pelas diversas plantas dos mais diversos tipos. Na Figura 1está selecionada uma vista que permite ter uma visão global sobre a quantidade total de energia a ser produzida, em tempo real. O mesmo sistema tem capacidade de monitorizar toda a

produção em toda a sua extensão geográfica, e ainda de diferentes tipos, como por exemplo eólica e solar.

Figura 1 – visão centralizada da produção energética em tempo real

A supervisão em tempo real oferecida pelo produto permite, através da leitura dados de diferentes plantas e locais (ambientes heterogéneos), perceber rapidamente quando ocorrem problemas com equipamentos através do módulo de alarmes. Quando ocorre um alarme, podem ser criados incidentes para a gestão do tratamento do elemento causador o problema, como ilustrado na Figura 2.

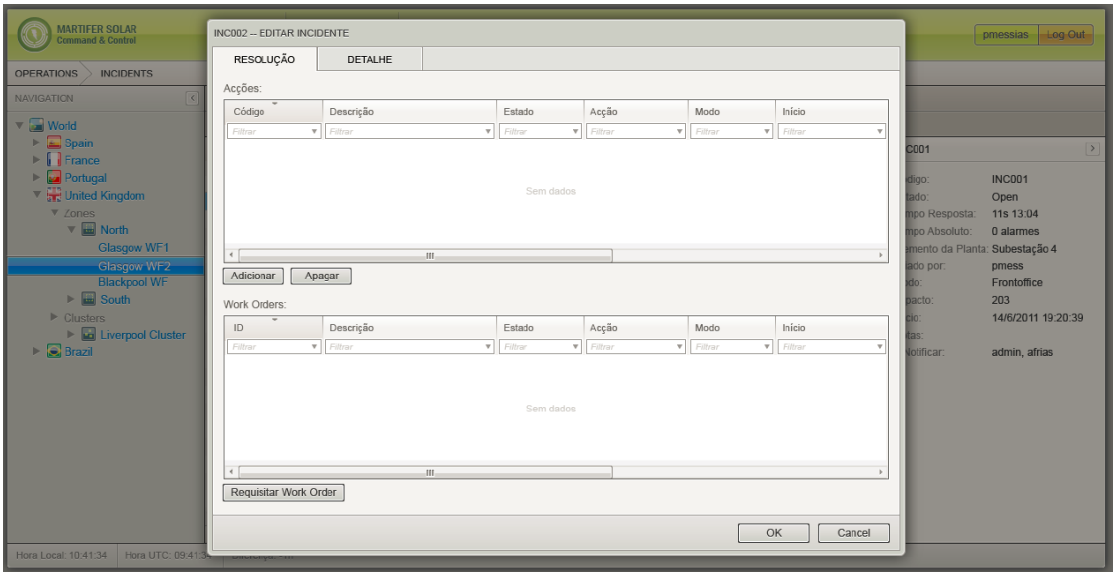


Figura 2 – gestão de incidentes

Para além de ser útil na componente reativa da interação com os equipamentos, este produto tem módulos capazes de fazer a gestão programada de planos de manutenção, isto é, manutenção preventiva. A Figura 3 ilustra o módulo de planos de manutenção, onde é possível gerir a manutenção preventiva de diferentes equipamentos de uma planta energética.

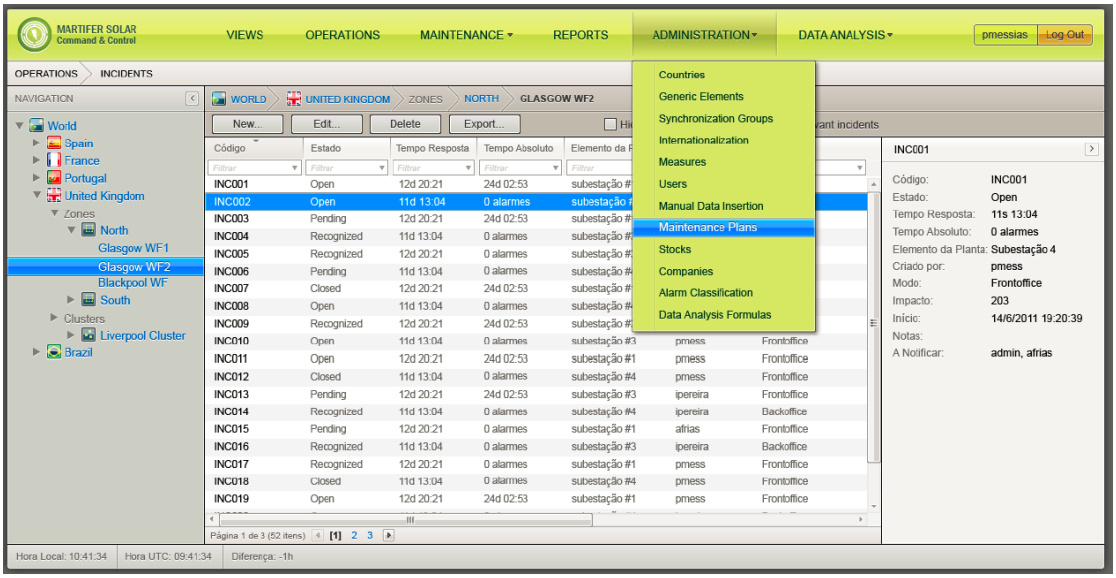


Figura 3 – planos de manutenção de equipamentos

Toda a informação necessária dos diferentes equipamentos das plantas configuradas no sistema EMS é recolhida por um componente, o DAM (Data Access Module), um serviço que acede a cada uma das localizações e lê em tempo real a informação disponibilizada pelos equipamentos. O DAM tem a capacidade de ler informação de equipamentos de diferentes tipos, disponibilizada por diferentes protocolos, convertendo-a num formato único para o sistema EMS, que é guardado numa base de dados centralizada. Este componente é configurável através do Portal Web, o que torna esta solução bastante flexível e adaptável a vários contextos.

O produto EMS já tem o sucesso comprovado, uma vez que se encontra instalada em diversos clientes e geografias, monitorizando um total de mais de 1GW de energia (Critical Software, 2014). Nos clientes onde se encontra implementado, o EMS garante:

- Gestão de mais 30% de ativos com os mesmos recursos
- Redução dos custos de operação e manutenção em 10%
- Diminuição do tempo de inatividade, aumentando a eficiência das plantas em 5%
- Maior controlo sobre os KPIs contratados

- Aumenta a capacidade de tomar decisões

1.2 A organização e o projeto EMS

A Critical Software (CSW) nasceu em Coimbra em 1998, como spin-off da incubadora de empresas do Instituto Pedro Nunes (IPN) da Universidade de Coimbra. É uma empresa de software e sistemas de informação, especializada na entrega de soluções fiáveis, serviços e tecnologias que suportam elementos críticos de negócio, como a segurança e proteção de indivíduos e equipamentos ou a garantia e fiabilidade em processos críticos, através do fornecimento de serviços, desenvolvimento de soluções à medida ou produtos de software capazes de satisfazer as necessidades dos seus clientes (Critical Software 2014).

A Critical Software atua no desenvolvimento de soluções de software à medida, consultadoria de gestão, *managed services* e desenvolvimento de produtos, área onde se insere o EMS.

O EMS é um projeto de produto na Critical Software, que garante não só o processo contínuo de correção de problemas e suporte a clientes, como também a implementação de novas funcionalidades que acrescentem valor ao produto e ainda o processo contínuo de otimização das funcionalidades já existentes. A metodologia de desenvolvimento utilizada é o Scrum[1].

À data da submissão da proposta que deu origem a este trabalho eu fazia parte da equipa de desenvolvimento do EMS, e o objetivo era este trabalho ser um estágio a realizar dentro da equipa de projeto. No entanto, por interesses maiores da Critical Software, fui deslocado para outro projeto, pelo que se tornou impossível realizar o trabalho no modelo pensado inicialmente.

Sendo assim este trabalho foi realizado sob a forma de projeto pessoal, tirando contudo partido do código fonte, a documentação de projeto e a infraestrutura existente, gentilmente cedidos pela Critical Software, e ainda da colaboração de recursos da empresa, em particular elementos da equipa do EMS, que foram sempre bastante prestáveis e colaborativos no sentido de tornar este trabalho possível.

[1] *Scrum* é uma metodologia de desenvolvimento de software ágil.

Há portanto que clarificar que existem dois projetos diferentes a decorrer: um desses projetos é o projeto pessoal, realizado por mim, que deu origem a este relatório. O outro projeto é o projeto EMS, a decorrer na CSW, relativo às tarefas de suporte e manutenção do produto e implementação de novos módulos e funcionalidades. O meu projeto pessoal, realizado fora da equipa de desenvolvimento do produto EMS, teve sempre em conta o funcionamento do projeto EMS na CSW, e a adequabilidade das soluções estudadas a esse mesmo contexto.

1.3 Motivação

Diversas razões motivaram este trabalho de estudo de evolução entre arquiteturas do EMS, nomeadamente:

- Melhorias no processo de codificação e testes;
- Melhorias no processo de desenvolvimento em Scrum;
- Melhorias ao nível do desempenho;
- Adaptação às necessidades de novos clientes.

1.3.1 Melhorias no processo de codificação e testes

Um projeto de software da dimensão do EMS é um projeto constantemente inacabado. Com cada nova funcionalidade existem defeitos inadvertidamente injetados, com a correção de cada bug há otimização que pode e deve ser feita, e com otimização pode surgir a necessidade de fazer um *refactoring*[2] ao código (Yakima, 2011).

Existem portanto atividades que são desenvolvidas em todo o decurso de um projeto de software, e que são as atividades que representam o maior volume de tempo despendido para execução do mesmo, nomeadamente codificação e testes.

A arquitetura de software implementada tem um impacto fundamental na execução eficiente dessas atividades. Uma arquitetura que facilite o processo de testes é crucial num produto, em particular num produto em constantes restruturação, derivada de

[2] *Refactoring* é o processo de adaptação do código a alterações estruturais. Por exemplo, quando se muda o nome de um método público e se têm de alterar todas as chamadas a esse método.

manutenção corretiva, evolutiva, e novas funcionalidades que vão sendo implementadas (Helloco, 2011).

Sistemas que podem facilmente ser testados podem facilmente ser alterados. Quanto menos esforço for gasto em testes e desenvolvimento, menor o custo de desenvolvimento do produto, o que irá permitir reduzir o preço e torná-lo atrativo para novos clientes. Uma das vantagens conhecidas do padrão de desenho MVC é a facilidade de implementação de testes, e essa é sem dúvida uma das razões de interesse na realização deste estudo.

1.3.2 Melhorias no processo de desenvolvimento em Scrum

O facto do produto EMS ser desenvolvido com uma metodologia de desenvolvimento ágil, nomeadamente o Scrum, aumenta o impacto que a arquitetura tem na eficiência de uma equipa de projeto.

Com *sprints*[3] de curta duração, duas semanas no caso deste projeto, o âmbito de cada uma delas tende a ser mais curto. Ao implementar por vezes módulos complexos, torna-se complicado encaixar as tarefas necessárias no tempo de duração de um *sprint*, e também nos elementos da equipa, isto é, por vezes é difícil distribuir tarefas do mesmo módulo por vários elementos de uma equipa. Em ASP.NET Web Forms existe um grande acoplamento ou grau de dependência entre os elementos de software, o que torna difícil o trabalho em equipa, nomeadamente a divisão de tarefas.

A divisão das responsabilidades do código em modelo, vista e controlador, promovida pelo MVC, promove um nível de abstração bastante mais completo, e no cenário em análise seria muito mais simples distribuir as tarefas de implementação por diferentes elementos de uma equipa, tirando inclusive partido das melhores capacidades de cada elemento (por exemplo, elementos mais fortes na apresentação podiam-se focar mais na parte das vistas).

Para além desta vantagem, o facto de ser utilizado o método Scrum dá um maior relevo à importância da arquitetura para os testes e manutenção do código. Trata-se de um modelo de desenvolvimento iterativo em que em cada *sprint* se tem de garantir a

[3] *Sprints* são as unidades básicas de calendarização de desenvolvimento em Scrum, tipicamente entre 1 a 4 semanas.

funcionamento de um determinado conjunto de *stories*[4], muitas vezes não considerando *stories* futuras, que quando forem implementadas irão requerer um grande *refactoring* do código implementado na *sprint* atual. É uma particularidade desta metodologia que tem a ver com a *definition of done*[5], que a torna bastante eficiente em desenvolvimento iterativo, mas que requer uma boa arquitetura e processo de validação, para que em cada *sprint* se garanta o correto funcionamento dos módulos e funcionalidades já entregues.

1.3.3 Melhorias ao nível do desempenho

O projeto EMS utiliza ASP.NET 4.0 para a implementação de toda a componente de Portal Web.

O ASP.NET tradicional, isto é, aquele que utiliza o paradigma Web Forms (e não o MVC) remonta a 2002, altura em que a Microsoft introduziu no mercado o ASP.NET 1.0. Esta tecnologia foi o caminho escolhido então pela Microsoft em termos de implementação Web, permitindo tirar partido de toda a Framework .NET, incluindo a possibilidade de implementar o código em linguagens distintas (C# ou VB.NET).

A abordagem da Microsoft para o desenvolvimento desta *framework* foi focada na facilidade de implementação de formulários Web, usando um modelo em muitos aspetos semelhante ao do desenvolvimento Windows Forms. O desenvolvimento de formulários consiste essencialmente na definição dos controlos e sua disposição, e na implementação de eventos associados a interação do utilizador sobre os mesmos (por exemplo, quando um utilizador clica num botão). Este modelo de programação abstrai os programadores de vários aspetos fundamentais das tecnologias que dão suporte ao desenvolvimento Web, incluindo o próprio HTML e o protocolo HTTP.

A capacidade de abstração dos protocolos utilizados pela tecnologia permite aos programadores, de uma forma simplificada, executarem operações que, com outras tecnologias, podem ser uma grande dor de cabeça. Por exemplo, a persistência de estado entre pedidos sucessivos.

[4] Stories são uma forma de descrever funcionalidades a implementar numa aplicação, onde é utilizada a perspetiva do utilizador final.
[5] Conjunto de critérios analisados para verificar se o desenvolvimento de um item pode ser dado como concluído.

As tecnologias Web assentam sob o protocolo HTTP, um protocolo *stateless*[6]. Entre pedidos sucessivos entre um cliente, tipicamente um *browser*, e o servidor, não existe qualquer persistência de informação. Isto se não usarmos, obviamente, a plataforma ASP.NET 4.0, Web Forms, uma tecnologia que nos permite quase que abstrair desta limitação. Existem vários recursos disponíveis pela tecnologia que permitem que, de forma automática, em cada pedido POST em que exista informação de um pedido ao servidor anterior, seja recuperado o estado dos objetos desde o momento desse pedido.

O mais comum será talvez o Viewstate[7], em que de forma completamente transparente ao utilizador, tirando partido de um *hidden field* no formulário HTML, é transportada informação entre pedidos que permitirão num POST posterior recuperar, no código servidor, o estado original dos objetos do pedido anterior. Este exemplo é talvez um dos mais ilustrativos do poder da tecnologia ASP.NET Web Forms na forma de simplificar aspetos mais complexos da tecnologia de suporte. No entanto existem contrapartidas associadas ao uso desta tecnologia. O exemplo do Viewstate é um caso típico, uma vez que quando usado em demasia afeta drasticamente o desempenho da aplicação, uma vez que o tamanho dos pedidos HTTP tende a crescer drasticamente, o que implica grande carga em termos de largura de banda e de processamento ao nível da serialização [8]dos objetos .NET em texto e processo inverso, na informação transportada entre cliente e servidor (Reed, 2006).

Como o Viewstate existem outros casos de recursos ASP.NET que podem afetar o desempenho, com o objetivo de simplificar o processo de implementação.

A este nível a arquitetura ASP.NET MVC adotou uma abordagem muito mais próxima do mecanismo por debaixo de uma aplicação web. A tecnologia elimina alguns dos facilitismos providenciados pelo ASP.NET Web Forms, mas ao mesmo tempo a sua abordagem muito mais próxima das tecnologias de suporte ao desenvolvimento Web permite um maior controlo sobre o que se implementa. É exigido aos programadores muitos mais conhecimentos do modelo de funcionamento e estes têm muito maior controlo sobre o que realmente é executado com cada bloco de código.

[6] Um protocolo *stateless* trata cada pedido como uma transação única, não relacionado de qualquer forma com o pedido anterior.
[7] Viewstate é um mecanismo da *framework* ASP.NET que permite manter o estado de variáveis ente pedidos http sucessivos.
[8] Serialização é o processo de transformação de um objeto em *bytes*, ou texto, de forma a poder ser persistido ou transportado numa aplicação.

Com esta tecnologia pretende-se, entre outros objetivos, perceber no projeto EMS até que ponto é possível melhorar o desempenho, uma vez que o desempenho do Portal Web é claramente um dos pontos a melhorar no produto.

1.3.4 Adaptação às necessidades de novos clientes

Sendo o EMS um produto cujo objetivo será vender ao maior número possível de clientes, existem particularidades da arquitetura MVC que poderão ser úteis no futuro.

Manter um produto que é vendido e usado por vários clientes é um desafio acrescido. No processo de desenho e conceção do produto deverá existir sempre um estudo sobre quais são as funcionalidades, e a forma de as implementar, que melhor dá resposta a um maior número de clientes.

O padrão de arquitetura MVC é uma mais-valia para um cenário deste tipo, na medida em que promove um baixo acoplamento entre os componentes de código responsáveis pelo modelo, vista e controlo. Existe mesmo a separação de conceitos, em que a vista, a lógica de negócio e o controlo de interação são completamente agnósticos e independentes entre si.

Esta particularidade poderá ser um fator decisivo na adequabilidade da solução a vários clientes. Por exemplo, o processo de adequação do Portal à imagem de uma empresa ou organização irá incidir sobretudo nos componentes vista do código implementado, ao mesmo tempo que a implementação de uma particularidade ao nível no negócio de um determinado módulo para uma empresa deverá incidir no componente modelo.

Este nível de independência entre os três conceitos permite por um lado uma muito maior assertividade e uma menor taxa de erros no momento da intervenção no código, e por outro lado permite mais facilmente manter vários ou diferentes níveis de comportamento, apresentação ou negócio para o mesmo produto, adequando-se a uma maior variedade de clientes.

1.4 Objetivos deste trabalho

Com este trabalho pretende-se comprovar a viabilidade de evolução de um projeto existente, implementado em ASP.NET 4.0, para ASP.NET MVC. É um processo complexo, para o qual não existe nenhum *template* nem nenhum guião, e como tal é um processo de descoberta.

Normalmente a questão ASP.NET ou ASP.NET MVC coloca-se antes da implementação de projetos, optando por uma das soluções possíveis, e são raras as referências bibliográficas ou na comunidade web sobre evoluções ou migrações em projetos existentes. No caso concreto do EMS e, acredito, em muitos outros projetos existentes implementados em ASP.NET, nos quais se perspetive um período longo de manutenção e evolução do produto, será do interesse dos responsáveis do projeto perceber se têm uma alternativa em termos de arquitetura, e existindo essa alternativa, qual o custo e o benefício de se proceder à transição ou migração.

O EMS é já uma solução tecnológica madura para a qual interessa apurar a viabilidade de evolução para MVC no decorrer do ciclo de vida do produto, isto é, mais do que reescrever o código todo do produto, interessa encontrar uma solução que encaixe no ciclo de vida do produto, sem prejuízos de relevo para os seus objetivos atuais. Mais do que se reescrever todo o código, solução claramente inviável por diversas razões, interessa encontrar uma abordagem possível para a evolução gradual do EMS para ASP.NET MVC, que permita que as das tecnologias coexistam no período de transição. Dessa forma a mudança de padrão de arquitetura poderá ser distribuída no tempo.

Para além de verificar a viabilidade do processo de evolução gradual da arquitetura do EMS, este trabalho tem ainda como objetivo definir algumas linhas orientadoras sobre a forma como essa migração pode ser concretizada. Deverá ser tida em consideração não só a evolução da arquitetura propriamente dita, mas também a capacidade de, durante este processo, garantir as atividades já programadas no ciclo de vida do produto, nomeadamente a manutenção corretiva e evolutiva.

1.5 Organização e estrutura do relatório

Este relatório divide-se nas seguintes secções.

No capítulo primeiro descrevem-se as notas introdutórias deste trabalho. O contexto em que se insere este trabalho, nomeadamente o projeto de produto em questão e contexto associado, e ainda as motivações que levaram ao interesse na realização deste trabalho, e que o tornam uma mais-valia.

No segundo capítulo, que resulta da análise técnica do EMS e tecnologias utilizadas, é feita em primeiro lugar uma descrição detalhada da tecnologia sob a qual assenta a arquitetura, para posteriormente ser descrita a própria arquitetura. A terceira parte do capítulo incide sobre a descrição de alguns problemas e oportunidades de melhoria neste projeto, em particular aqueles em que se calcula que uma evolução para o padrão MVC pode ser uma vantagem.

O terceiro capítulo é focado na análise da nova tecnologia, ASP.NET MVC, descrevendo o novo modelo da Microsoft, em particular nos pontos em que é estrategicamente diferente do ASP.NET tradicional. Tendo em conta essas diferenças e os problemas e oportunidades identificados, serão neste capítulo também descritas alterações propostas à solução atual do EMS para adequação a esta nova tecnologia, e os benefícios potenciais que se perspetivam da evolução do modelo atual.

No quarto capítulo é descrito o processo de implementação de uma prova de conceito, que consiste em montar o ambiente de desenvolvimento necessário para integrar as tecnologias, quer as relativas à solução existente, em ASP.NET Web Forms, e aquelas da nova arquitetura proposta, nomeadamente ASP.NET MVC. Uma vez montado esse ambiente foram executados trabalhos no sentido de avaliar a viabilidade quer da criação de novos módulos em MVC, quer da migração dos existentes para a nova arquitetura.

No quinto capítulo é feita uma análise aos resultados obtidos, nomeadamente uma visão crítica sobre a forma como decorreram os trabalhos de elaboração da prova de conceito.

O sexto capítulo relata as conclusões retiradas da realização deste projeto, nomeadamente tendo em conta a concretização ou não dos objetivos propostos.

O relatório termina com a bibliografia de referência para a realização deste trabalhos e os anexos.

2 Análise da solução tecnológica atual

O primeiro passo a executar antes de considerar mudar as tecnologias e arquitetura é conhecer as tecnologias e arquitetura atuais. Nesse sentido este capítulo documenta a tecnologia atual do EMS que se pretende evoluir, o ASP.NET, em particular na forma como encaixa na arquitetura de software do portal web do sistema EMS.

A análise relativa às tecnologias atuais e arquitetura irão incidir apenas sobre o componente do sistema em análise, o Portal Web. Para uma visão mais abrangente do sistema, como ilustrado na Figura 4, por favor consultar o Anexo A: documento de arquitetura do EMS.

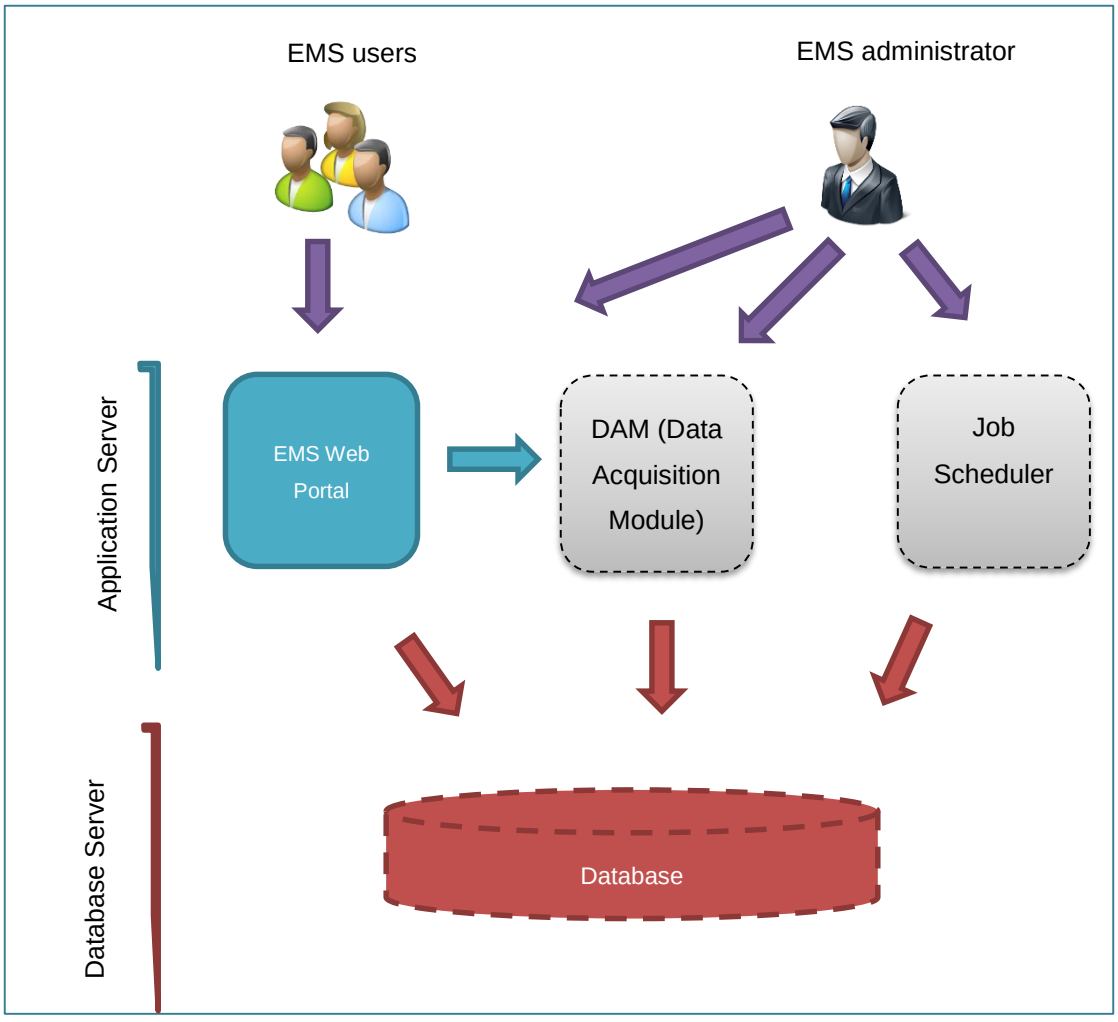


Figura 4 – principais componentes de software do EMS

2.1 Tecnologia ASP.NET

O Portal Web, um dos principais componentes de software do EMS, encontra-se implementado com recurso à tecnologia ASP.NET 4.0.

2.1.1 Origem e evolução do ASP.NET

A primeira versão da *framework* ASP.NET, a versão 1.0, remonta a 2002. Na altura a Microsoft introduziu esta tecnologia com apenas um modelo de desenvolvimento, chamado Web Forms (o modelo utilizado no EMS).

O modelo Web Forms foi introduzido como uma forma rápida e intuitiva de desenvolver aplicações web, não muito diferente ao modelo utilizado já pela Microsoft para desenvolver aplicações Windows.

O ASP.NET veio substituir o ASP clássico, uma metodologia que misturava nos mesmos ficheiros de código HTML com código de servidor, que era interpretado, e não compilado como acontece atualmente. A versão 1.0 da tecnologia foi a primeira a introduzir o conceito de separação das páginas em dois ficheiros:

- Um ficheiro de extensão aspx onde se define com uma sintaxe XML própria a definição da interface;
- Um ficheiro de extensão .aspx.cs ou aspx.vb, dependendo da linguagem .NET utilizada (C# ou Visual Basic .NET), para a implementação do código relativo a eventos.

Desde o lançamento da primeira versão do ASP.NET que têm surgido diversas revisões da tecnologia, com melhorias significativas. Em 2005 surgiu o ASP.NET 2.0, com melhorias funcionais ao nível dos controlos disponíveis, as Master Pages[9], Themes [10]e Skins[11], e muitas outras funcionalidades. Um ano mais tarde chega a versão 3.0, que acrescenta à versão anterior integração com quatro tecnologias Microsoft, nomeadamente Windows Comunication Foundation (WCF)[12], Windows Presentation

[9] *Master Pages* são um artefacto ASP.NET que possibilita a definição de um layout base que se repete num conjunto de páginas e que apenas necessita de ser implementado uma vez

[10] *Themes* em ASP.NET consistem na definição de um conjunto de valores atribuídos a propriedades para um determinado tipo de páginas e controlos, permitindo que a definição desses valores seja consistente em toda a aplicação

[11] *Skins* são a definição de estilo individual de um tipo de controlo ASP.NET.

[12] WCF (Windows Comunication Foundation) é uma framework Microsoft .NET para aplicações orientadas a serviços.

Foundation (WPF)[13], Windows Workflow Foundation (WF)[14] e Windows Cardspace[15]. Com a versão 3.5 foram adicionados novos controlos à *framework*, e ainda foi integrado o ASP.NET AJAX [16]na plataforma, entre outras funcionalidades. Com a versão 4.0 foi melhorado o Output Cache, foi introduzido o JQuery [17]como a libraria Javascript [18]por omissão, foi melhorado o Viewstate, entre outras funcionalidades. Entretanto surgiu também a versão 4.5, que tem como principais novidades operações HTTP assíncronas, HTML 5 e muitas outras funcionalidades.

O EMS usa a versão 4.0 da *framework* .NET.

Desde então surgiu um paradigma alternativo ao ASP.NET Web Forms: o ASP.NET MVC. Este novo paradigma não deve ser visto como um substituto do anterior, pois possui caraterísticas próprias, com as suas vantagens e desvantagens. O modelo de programação Web ASP.NET Web Forms não ficou obsoleto, recebendo constantes atualizações e melhorias implementadas pela Microsoft, com total suporte e integração com as versões mais recentes do IDE [19]Visual Studio, e continua a ser muito utilizado no mundo do desenvolvimento de software (Vogel, 2013).

2.1.2 Modo de funcionamento

Em ASP.NET para cada página existem dois ficheiros. Um ficheiro XML de extensão aspx, com uma sintaxe própria onde é colocada a definição da página ao nível da disposição e configuração dos controlos. Esse ficheiro é sempre acompanhado de um ficheiro de extensão aspx.cs ou aspx.vb, dependendo da linguagem de programação do *code behind*, C# ou Visual Basic .NET, onde é implementado o código relativo a eventos da página ou dos controlos, que permitem implementar determinados comportamentos ou ações.

[13] WPF (Windows Presentation Foundation) é um sistema gráfico para desenho de interfaces em Microsoft .NET
[14] WF (Windows Workflow Foundation) é uma framework que possibilita a escrita de processos de sistemas e humanos em programas de software
[15] Windows Cardspace é um componente de software vocacionado para integrar a identidade digital de utilizadores de forma simplificada
[16] AJAX é a interligação de um conjunto de tecnologias utilizadas na componente cliente de aplicações web para implementação de comportamentos assíncronos
[17] JQuery é uma libraria Javascript para manipulação de elementos HTML, eventos, animações e AJAX
[18] Javascript é uma linguagem orientada a objetos utilizada para obter efeitos interativos em *browsers*
[19] IDE – *integrated development environment*, ou ambiente integrado de desenvolvimento. É o software que disponibiliza as ferramentas necessárias para o desenvolvimento de software.

Cada página e respetivos controlos em ASP.NET permitem implementar lógica condicional em eventos, implementados em código .NET. Cada vez que um evento, como por exemplo o clique de um botão, é acionado, o código implementado é executado, e consequentemente a ação desejada é concretizada. Não só os controlos, como botões, permitem definir código em eventos. Cada página em ASP.NET tem um ciclo de vida próprio cada vez que é feito um pedido HTTP via navegador cliente. Existe um conjunto de eventos, desde o início do pedido até que o mesmo esteja processado, que pode e é habitualmente utilizado para implementar comportamentos. Um exemplo típico é o evento Page_Load, que ocorre quando cada página é carregada, e que tipicamente é utilizado para carregar da base de dados os dados necessários para o processamento da página.

Em ASP.NET o programador abstrai-se de conceitos básicos intrínsecos à tecnologia web, como por exemplo o próprio HTML ou o protocolo HTTP. O ASP.NET foi desenhado para ser trabalhado como uma linguagem orientada a objetos, em muitos aspetos semelhante à programação Windows Forms, e é sobretudo com objetos que trabalha o programador. Esses objetos, próprios da *framework* ASP.NET Web Forms, irão implementar toda a lógica relativa às tecnologias mais baixo nível. O conhecimento do funcionamento de cada uma destas classes, embora não seja fundamental para se poder implementar uma página ASP.NET, é altamente recomendado para que o resultado do trabalho realizado seja sustentável num projeto de maiores dimensões.

Repare-se no seguinte exemplo da Figura 5: uma pequena página web de demonstração com um botão e um texto, em que quando se clica no botão é escrita no ecrã a hora atual. Na definição do *markup* [20]são colocados elementos que não são HTML, mas sim a definição de controlos ASP.NET.

```
<form id="form1" runat="server">
<div>
 <asp:Button ID="Button1" runat="server" Text="Button"
     OnClick="Button1_Click" BackColor="Blue" ForeColor="Yellow" />
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</div>
</form>
```

Figura 5 – markup de página exemplo

[20] Markup é uma forma de definir, usando um conjunto conhecido de tags textuais, uma forma de apresentação da informação numa página.

Nestes controlos ASP.NET podem ser registados eventos que respondem, por exemplo, a uma ação do utilizador. Neste caso, ao clicar no botão é executado um bloco de código no servidor que irá mostrar a caixa de texto com a data atual. Existe outros eventos que são registados de acordo com o ciclo de vida da página, por exemplo, sempre que a página é carregada, como ilustrado na Figura 6.

```
public partial class Page1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            TextBox1.Visible = false;
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        TextBox1.Visible = true;
        TextBox1.Text = DateTime.Now.ToLongTimeString();
    }
}
```

Figura 6 – o code behind da página de exemplo

Na prática, o motor do ASP.NET irá interpretar as alterações feitas aos objetos ASP.NET e criar o HTML correspondente. Para o exemplo de *markup* da Figura 5 é gerado o HTML correspondente, ilustrado na Figura 7.

```
<div>
 <input type="submit" name="Button1" value="Button" id="Button1"
 style="color:Yellow;background-color:Blue;" />
    <input name="TextBox1" type="text" value="23:03:58" id="TextBox1" />
</div>
```

Figura 7- HTML gerado pelos controlos ASP.NET

Existe no entanto uma particularidade do ASP.NET Web Forms que é essencial referir. Sendo o protocolo HTTP sobre o qual é transportada esta informação um protocolo *stateless*, é utilizado um recurso, o Viewstate, que de forma transparente para o utilizador guarda a informação na página HTML que, ao ser passada ao servidor no pedido seguinte, lhe irá permitir reconstituir o estado dos objetos, nomeadamente os

controlos, que foram criados ou utilizados no pedido anterior. Essa informação é enviada no pedido HTTP sob a forma de um *hidden field* com um tratamento especial pela plataforma, como ilustrado na Figura 8.

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="+FlIwA+DiAE7wjtztEpWrNmTtcvmxdSCD5+LDjkOwkb1sKRw+
b2Ih+u9wr7KA/gn0/3m0d/+IdUoYNoz52mSVI4tWy6L2vUovHcoWR+ZHB2Ho1cKZi4WIK8ATXwBBRkl" />
```

Figura 8 – hidden field gerado pelo Viewstate

Desta forma sem qualquer acréscimo de trabalho de codificação para o programador, quando o formulário for submetido para o servidor, ao clicar no botão, propriedades como por exemplo a cor do botão existirão da mesma forma como o controlo foi contruído. É este um dos principais atributos do modelo ASP.NET Web Forms que o torna um modelo de programação Web bastante simples e intuitivo, não muito diferente da programação orientada a objetos. Existem no entanto inconvenientes relacionados com este recurso, relacionados sobretudo com o desempenho, como iremos aprofundar ainda neste relatório.

2.1.3 Estruturação das páginas em ASP.NET

A *framework* ASP.NET possibilita aos programadores estruturarem o conteúdo visual dos sites de forma eficiente. Componentes que se repetem em todas as páginas não têm de ver a sua implementação repetida mais do que uma vez. Páginas complexas podem também ser divididas em componentes mais pequenos, de forma a relativizar a sua complexidade.

Nas tarefas relacionadas com a evolução tecnológica deverão tirar partido desta estruturação para análise na prova de conceito, pelo que interessa debater este assunto.

Os principais componentes que estruturam um site em ASP.NET são os seguintes:

As *master pages* são os elementos da página que integram os componentes que se repetem em todas ou na grande generalidade das páginas. Por exemplo o logotipo, o menu, o rodapé. Cada *site* pode conter mais do que uma *master page*, sendo que em cada página é indicada a *master page* que a complementa. Podem existir vários níveis de *master pages*, constituídos por uma *master page* base e uma ou mais *master pages*,

que complementam alguma funcionalidade à anterior, implementando uma forma de herança visual e por vezes comportamental.

As páginas são os principais elementos que compõem um site, e são constituídas por controlos que geram HTML para o *browser* cliente. As páginas podem conter todos os controlos que a constituem ou podem complementar uma *master page*. Cada página tem um endereço único que possibilita ao utilizador navegar entre as diferentes páginas de um *site*.

Os controlos são blocos mais pequenos, que podem conter outros controlos dentro de si. Têm uma funcionalidade ou componente visual própria e muitas vezes são autossuficientes a realizar essa funcionalidade. Podem existir mais do que uma vez numa página ou em páginas diferentes.

2.2 Arquitetura atual do EMS

Uma análise completa da arquitetura atual do sistema sob a qual será feita a análise documentada neste relatório é um passo essencial para os objetivos deste projeto. Interessa identificar as partes ou componentes do sistema que deverão ser convertidos para o modelo ASP.NET MVC para que possamos retirar as necessárias conclusões.

Nesse sentido, a Figura 9 ilustra os principais componentes que constituem o portal EMS.

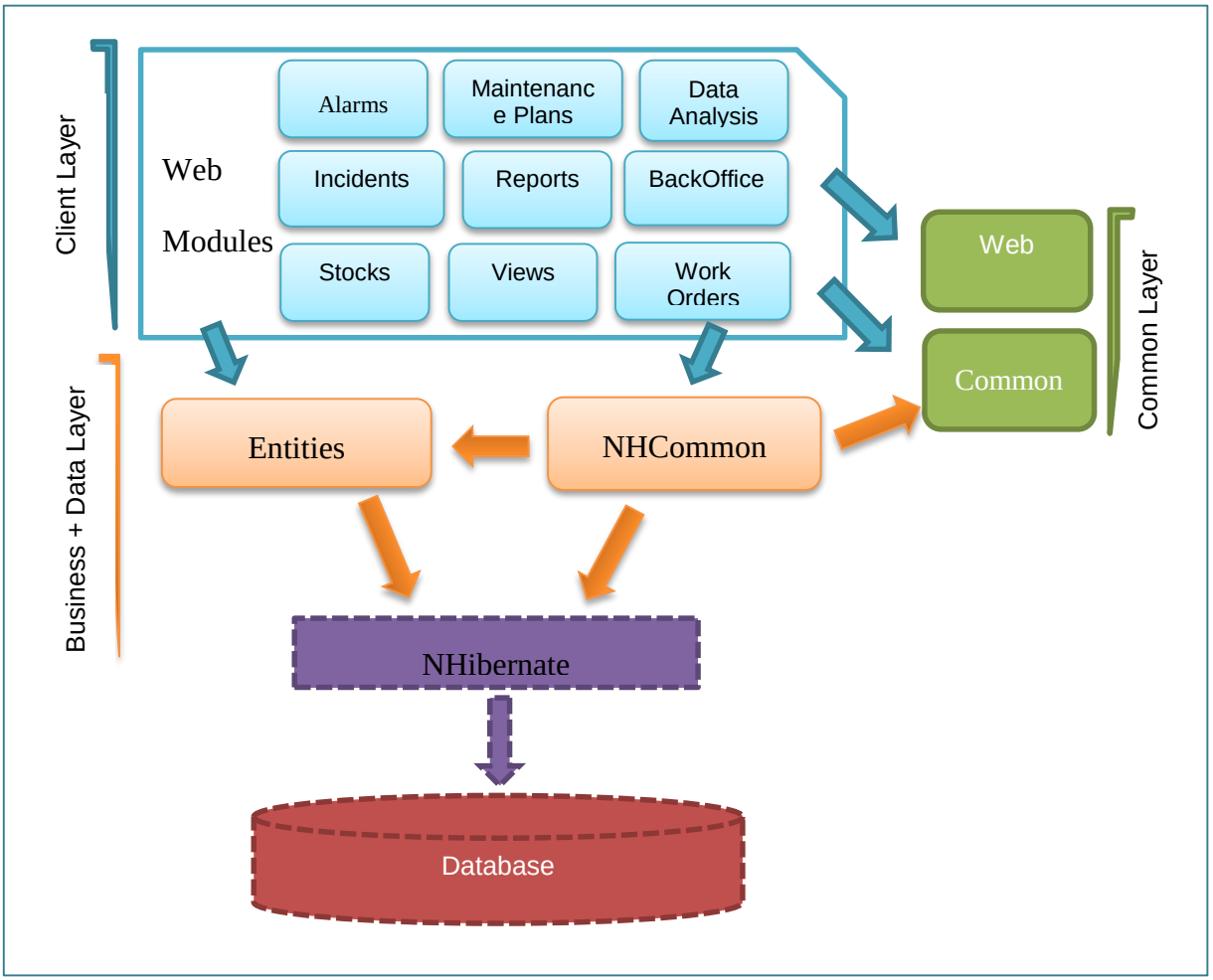


Figura 9 – visão da arquitetura do Web Portal

Para melhor perceber a aplicação no seu todo, iremos descrever cada um dos seus componentes.

2.2.1 Decomposição em camadas

A arquitetura do Portal pode ser decomposta nas seguintes camadas, como ilustra a Figura 9:

- A camada de negócio e dados implementa a lógica que transporta os dados da base de dados até às classes de negócio e vice-versa. Nas classes de negócio são implementadas as principais regras de negócio para que, quando a camada cliente invocar um método, os dados já tenham chegado trabalhados segundo os requisitos definidos.

- A camada comum consiste num conjunto de classes que são conhecidas por todas as camadas da aplicação e que definem os tipos de dados conhecidos quer pelas classes da camada cliente, quer pelas classes da camada servidor.
- A camada cliente implementa os formulários de interface com o utilizador. É esta camada que permite ao utilizador interagir com o sistema, fornecendo informação e consultando dados da base de dados (através de chamadas à camada de negócio e dados). Os formulários implementados nesta camada são agrupados em módulos, como ilustrado na Figura 9.

A camada cliente da arquitetura é aquela que terá maior importância para este estudo, uma vez que é esta que se encontra implementada sobre a *framework* ASP.NET Web Forms.

2.2.2 Decomposição da camada cliente em artefactos ASP.NET Web Forms

Em termos lógicos a camada cliente divide-se em diversos módulos, uma divisão cujo critério é a afinidade em termos de negócio, que por sua vez corresponde quase sempre a alguma afinidade em termos de formulários e outros componentes de código.

No entanto em termos técnicos já foram descritas neste relatório algumas particularidades da tecnologia ASP.NET Web Forms, que permitem estruturar a implementação das páginas eliminando a duplicação de código e diminuindo a complexidade, através da divisão em componentes de menor dimensão.

O Portal EMS tira partido dos principais artefactos do ASP.NET Web Forms a este nível. Esses artefactos e a forma como se relacionam entre si é ilustrada pela figura seguinte.

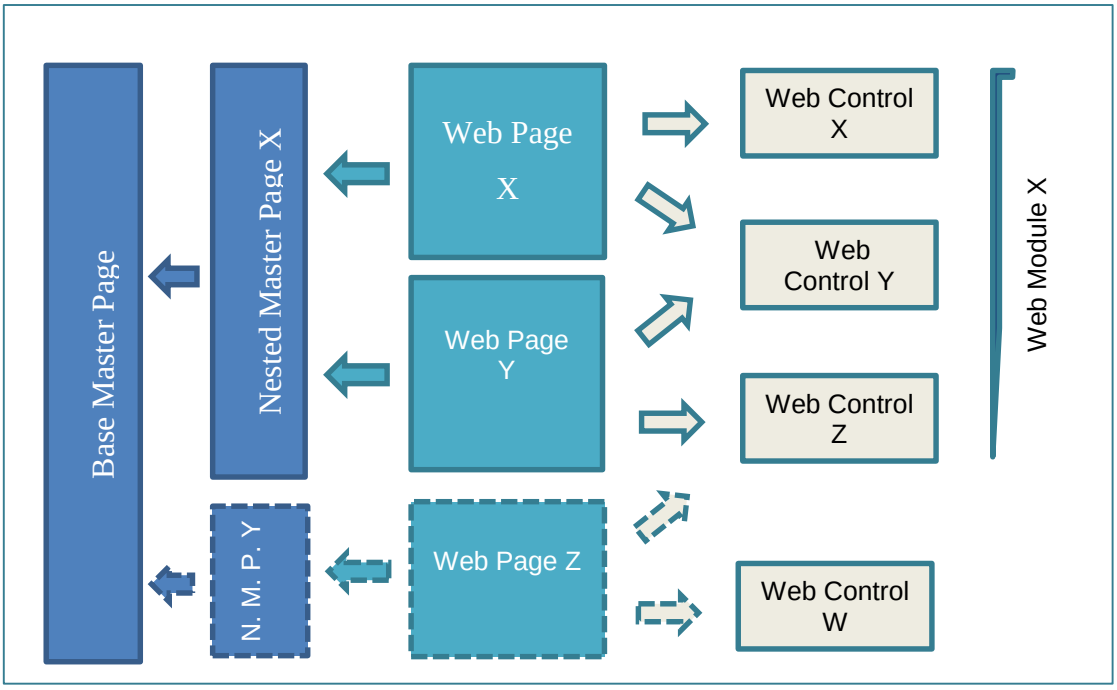


Figura 10 – estruturação de um módulo

Existe uma *master page* comum à generalidade dos módulos, mas podem existir outras que descendem da primeira, em que cada uma delas acrescenta elementos à estrutura, implementando herança visual. Cada módulo tem ainda uma ou mais páginas, que complementam os elementos comuns da *master page* com os formulários específicos da página em questão. Essas páginas podem conter *web controls*, controlos que implementam blocos de funcionalidade contidos, que são reutilizáveis.

2.3 Problemas e oportunidades

A tecnologia ASP.NET Web Forms é amplamente utilizada no desenvolvimento de software dos dias de hoje, consistindo, por isso, numa solução válida e bastante atual. No entanto, embora seja uma solução com alguns pontos fortes, como por exemplo a facilidade e a rapidez do desenvolvimento, tem também alguns pontos fracos que se tentarão minimizar na evolução para a nova *framework*.

2.3.1 Desempenho

Alguns recursos do ASP.NET Web Forms, como o exemplo do Viewstate, apesar de constituírem uma enorme mais-valia sob o ponto de vista da simplicidade e rapidez de desenvolvimento, têm contrapartidas por vezes drásticas ao nível do desempenho, sobretudo se não for usado de forma consciente.

Por vezes usa-se e abusa-se do Viewstate como forma de guardar informação necessária entre pedidos HTTP sucessivos, e isso tem um grande impacto ao nível do desempenho. A serialização do objeto Viewstate (processo de transformação de objetos em texto) consome alguns ciclos de processamento mas, pior do que isso, o facto de o Viewstate ser convertido num *hidden field* que transita entre *browser* e servidor dentro do pedido HTTP, faz com que o tamanho dos pedidos cresça imenso, tornando-os lentos e consumistas de largura de banda. A Figura 11 ilustra com um exemplo do EMS o peso que o Viewstate pode ter no volume de dados dos pedidos. O *hidden field* correspondente à informação do Viewstate tem um volume de dados com claro peso na funcionalidade.

Figura 11 – parte da serialização do Viewstate no hidden field (módulo stocks)

A migração de um módulo de ASP.NET Web Forms para ASP.NET MVC é uma oportunidade para rever este tipo de problemas.

2.3.2 Separação de conceitos

A arquitetura do ASP.NET Web Forms promove a separação do *markup* e do *code-behind*[21]. No entanto, é por vezes difícil desassociar a lógica de negócio do código responsável pela definição das páginas (Strahl, 2007). Existem elementos visuais que são por vezes implementados no *code-behind*, e também elementos comportamentais que ficam associados ao código XML do *markup*. A Figura 12 ilustra um exemplo concreto do EMS onde no mesmo método de *code-behind* é utilizado para executar logica de negócio, como por exemplo o carregamento de informação das classes de dados, ao mesmo tempo que manipula diretamente objetos da vista.

```
private void Populate()
{
    Reset();// Cleanup state.
    ASPxEdit.ClearEditorsInContainer(newTransferPopupDialog);// Cleanup validations.

    // Request input data.
    var e = new DataEventArgs();
    InputDataCallback(this, e);

    // Store input data in session.
    BasePageSession.Session[MovimentationIdSessionKey] = e.MovimentationId;

    // Populate form.
    var mov = movDAO.Load(e.MovimentationId);
    var itemType = mov.ItemType;
    var cls = itemType.Class;
    var subfamily = cls.Subfamily;
    var family = subfamily.Family;

    var stockQuantity = movDAO.GetExistenceCount(
        mov.Warehouse.WrhId, mov.ItemType.ItmId, mov.ItemOwner.ItoId,
        mov.MovState, mov.MovSerialNumber);

    familyTextBox.Text = family.Translation;
    subfamilyTextBox.Text = subfamily.Translation;
    classTextBox.Text = cls.Translation;
    itemTypeTextBox.Text = itemType.ItmDescription;
    codeTextBox.Text = itemType.ItmCode;
    typeNameTextBox.Text = CodeToTextHelper.GetItemTypeTypeText(itemType.ItmType);
    stockPowerPlantTextBox.Text = mov.Warehouse.PowerPlant.PwpName;
    stockWarehouseTextBox.Text = mov.Warehouse.WrhName;
```

Figura 12 – manipulação de elementos visuais no code-behind (módulo stocks)

[21] *Code-behind* na tecnologia ASP.NET corresponde ao código implementado em C# ou VB.NET que é implementado para executar os eventos dos objetos definidos no *markup*.

A inexistência de fronteiras bem definidas entre os conceitos é uma limitação da arquitetura com impacto significativo na manutenção do código do projeto. Qualquer problema que surja é mais difícil de detetar e corrigir, e o *refactoring* é mais arriscado.

Para além disso, é mais difícil atribuir tarefas aos elementos da equipa de projeto, em particular se pretendermos concentrar as pessoas onde são mais fortes. Por exemplo, um *designer* terá sempre de contribuir com *code-behind*, e um especialista nos requisitos e lógica de negócio dificilmente poderá realizar esse trabalho sem editar o *markup*. Pelos mesmos motivos é complicado colocar os recursos a trabalhar em conjunto.

2.3.3 Testabilidade

O código produzido nos artefactos típicos do ASP.NET Web Forms são difíceis de testar, em particular se pretendermos automatizar testes unitários, algo que num projeto com estas caraterísticas é indispensável (Strahl, 2007).

Qualquer elemento Web Forms, quer seja uma *master page*, uma página ou um controlo, irá misturar na mesma classe objetos, variáveis e lógica relativos a componentes visuais com os mesmos tipos de elementos mais vocacionados para as regras de negócio. Desta forma não é simples implementar testes unitários para testar blocos de lógica mais aptos a testes, como por exemplo as regras de negócio.

Ainda relativamente ao exemplo da Figura 12, implementar testes unitários para esse método seria complicado, uma vez que são manipulados objetos da vista. Seria mais simples isolar a lógica de negócio, e dessa forma poderíamos facilmente ter testes unitários para a lógica de negócio.

3 Proposta de nova solução tecnológica

A proposta de nova solução tecnológica surge na sequência da análise da solução atual do EMS baseada em ASP.NET Web Forms e da análise das potencialidades da tecnologia ASP.NET MVC, com o intuito de aplicar recursos desta nova tecnologia no projeto que possam revelar-se vantajosos para os problemas e oportunidades em foco.

Tal como já referido, para este trabalho o âmbito será a realização de uma prova de conceito, isto é, não será montada uma solução de raiz, mas sim encontrada uma solução de evolução que permita criar novos módulos utilizando esta tecnologia, capazes de coexistir com os existentes, que por sua vez poderão ser migrados de forma gradual. O que se pretende é provar a possibilidade de realização de uma evolução entre tecnologias, permitindo a coexistência entre módulos em ASP.NET Web Forms e MVC, de forma completamente transparente para o utilizador. É esta a forma mais praticável de realizar tal migração na fase atual do ciclo de vida do produto em análise, avaliando o seu impacto no desempenho global.

3.1 Tecnologias

A tecnologia em análise nesta secção do relatório é o ASP.NET MVC.

3.1.1 Origem e evolução do ASP.NET MVC

A *framework* de desenvolvimento ASP.NET MVC foi introduzida pela Microsoft em 2009, resultando da combinação da maturidade da base ASP.NET com as vantagens do padrão de desenho MVC. Desta forma, a Microsoft passou a oferecer uma solução alternativa ao modelo ASP.NET Web Forms, que melhora alguns aspetos, como a testabilidade, o controlo do HTML gerado, suporte para o desenvolvimento de grandes equipas em grandes projetos (através da separação dos conceitos de modelo, vista e controlo) e melhor manutenção do código produzido.

A primeira versão da *framework*, de 2009, introduz funcionalidades tais como o motor de *routing*[22], *helper methods* [23]para criar HTML e AJAX, *data binding*, o *Web Forms view engine*[24] e outros elementos core da tecnologia.

Em Março 2010 surgiu a segunda versão da *framework*, introduzindo um conjunto significativo de melhorias, com destaque para *templates* customizáveis, controladores assíncronos, o conceito de áreas, capacidade de separação de aplicações de grande escala em diferentes projetos, melhores ferramentas e integração com o Visual Studio.

Menos de um ano depois a Microsoft lança o ASP.NET MVC 3, com destaque para a introdução do novo *view engine* Razor, como alternativa ao *view engine* ASP.NET.

A versão 4 data de 2012, e é a versão que será utilizada neste projeto. Esta versão contempla como principais inovações tecnológicas o ASP.NET Web API, uma nova *framework* para contruir serviços HTTP RESTful[25], novos *templates* Visual Studio para HTML 5 e aplicações web *mobile*, suporte para a implantação de aplicações MVC na plataforma Microsoft Windows Azure[26], modelos de autenticação OAuth, OpenID ou Facebook, entre outras.

É de salientar o facto de que, desde a introdução deste modelo e *framework* de desenvolvimento, a Microsoft lançou simultaneamente várias versões da plataforma ASP.NET Web Forms. Inclusive, as versões ASP.NET MVC que foram lançadas foram tirando partido da integração de muitas dessas funcionalidades na plataforma ASP.NET base, partilhadas pelas duas *frameworks*.

[22] O motor de *routing* do ASP.NET MVC é o mecanismo responsável pelo mapeamento dos URL nos controladores e *action methods* correspondentes
[23] Os *helper methods* em ASP.NET MVC são métodos de classes específicas que servem para realizar tarefas recorrentes de forma simplificada, por exemplo, escrever um botão em HTML
[24] O *view engine* é um mecanismo de plataforma capaz de interpretar uma sintaxe própria e traduzir a mesma no HTML que é enviado para o *browser*
[25] REST (*Representational State Transfer*) é uma abstração no *World Wide Web* que possibilita o acesso a dados e funcionalidade através de um simples URL. Serviços *RESTFul* são serviços que implementam este paradigma.
[26] Windows Azure é uma infraestrutura de *cloud computing* da Microsoft

3.1.2 Modo de funcionamento

A *framework* ASP.NET MVC assenta sobre a *framework* base ASP.NET, e como tal componentes como segurança, manutenção de estado, autenticação e autorização, *caching* e muitos outros componentes são partilhados com a Framework Web Forms.

A grande novidade desta nova *framework* é a estruturação da informação de forma a separar a lógica de negócio da lógica de apresentação. Esta separação é conseguida através do isolamento lógico entre o controlador, o modelo e a vista.

O seguinte diagrama ilustra genericamente a forma como se relacionam entre si o modelo, a vista e o controlo. Cada um dos três conceitos será descrito em detalhe de seguida.

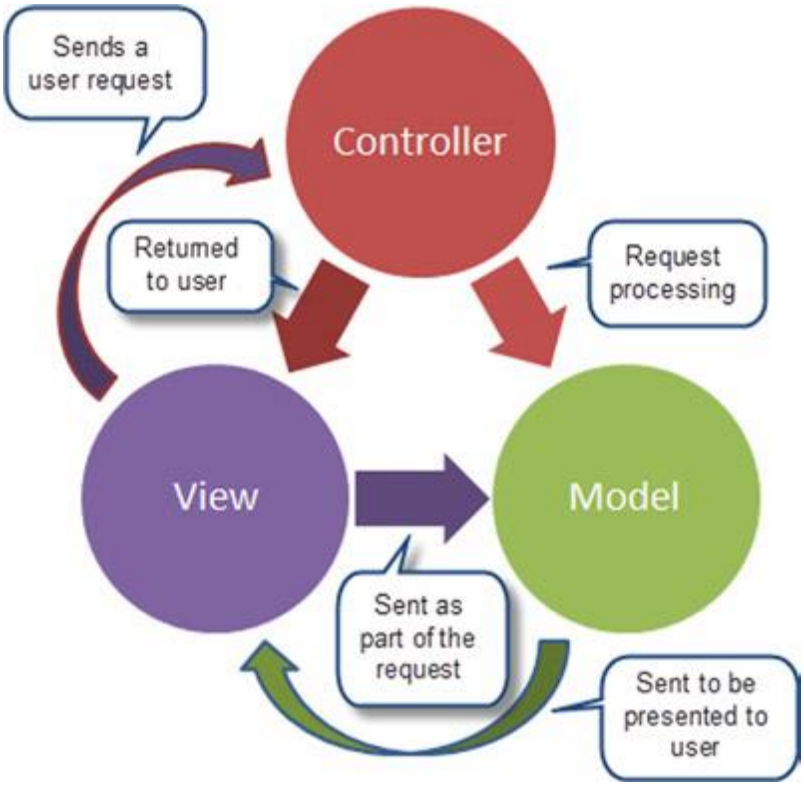


Figura 13 – representação da relação entre modelo, vista e controlador em MVC (Paz,2013)

3.1.2.1 Controlador

A função do controlador é o processamento de pedidos do utilizador. Cada pedido HTTP é executado por um controlador específico.

Cada classe controlador é derivada da classe base *Controller*, e implementa um conjunto de métodos, os *action methods*, que, como o nome indica, implementam ações que o site realiza através do pedido HTTP correspondente. No exemplo ilustrado na Figura 14, o controlador HomeController implementa três diferentes *action methods*, portanto terá capacidade para dar resposta a três tipos diferentes de pedidos, especificados por três diferentes padrões de URL.

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
```

Figura 14 – exemplo de classe controlador

O tipo de retorno de cada um dos *action methods* é essencial para a implementação do comportamento esperado na aplicação. No exemplo da Figura 14 todos os *action methods* devolvem ao *browser* uma vista, o que significa que essa vista irá gerar HTML que será interpretado e mostrado ao utilizador. No entanto, o tipo de retorno de um *action method* pode ser um conjunto de classes derivadas de *ActionResult*, como por exemplo: *FileResult*, para devolver ficheiros ao *browser* ou aplicação cliente; *JavaScriptResult* para devolver um script para ser interpretado pelo *browser* cliente; *RedirectResult* que será interpretado pelo cliente para fazer um novo pedido para um novo URL; *HttpUnauthorizedResult* para dar feedback sobre um acesso não autorizado; entre muitos outros que permitem implementar o tipo de comportamento das aplicações *web based* mais modernas.

O mapeamento de cada pedido na classe controlador e *action method* correspondentes é levada a cabo pelo *routing engine*. O *routing engine* é um mecanismo do ASP.NET MVC que é executado no momento do arranque da aplicação no servidor, e cria um conjunto de mapeamentos entre padrões no URL dos pedidos executados e os elementos MVC correspondentes, neste caso a classe do controlador, o nome do *action method* e ainda parâmetros.

No exemplo ilustrado na Figura 15 é definido um padrão de URL em que, para além do domínio, seguem-se o elemento relativo ao controlador, seguido de um elemento que identifica o *action method*, seguido de um parâmetro opcional.

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

Figura 15 – exemplo de roteamento para um controlador

3.1.2.2 Vista

No padrão de desenho MVC o elemento vista é o responsável pela apresentação de resultados do pedido ao utilizador. No exemplo do controlador do tópico anterior, quando um *action method* devolve um *ActionResult* do tipo *ViewResult*, é criado o objeto do tipo vista correspondente, que por sua vez irá gerar o HTML necessário e enviar de volta ao cliente que efetuou o pedido.

Numa aplicação web moderna, a apresentação de resultados não se limita à definição estática de modelos HTML. É necessário executar lógica de servidor condicional de forma a ajustar o formulário ao dinamismo que requer cada *site* ou aplicação. Para esse efeito, a plataforma ASP.NET MVC fornece aos programadores dois motores de geração de vistas:

- ASP.NET *view engine* – introduzido na primeira versão do ASP.NET, tem uma sintaxe muito semelhante à do ASP.NET Web Forms, como é visível no exemplo da Figura 16.

```
<asp:Content ID="errorTitle" ContentPlaceHolderID="TitleContent" runat="server">
    Error - My ASP.NET MVC Application
</asp:Content>

<asp:Content ID="errorContent" ContentPlaceHolderID="MainContent" runat="server">
    <hgroup class="title">
        <h1 class="error">Error.</h1>
        <h2 class="error">An error occurred while processing your request.</h2>
    </hgroup>
</asp:Content>
```

Figura 16 – exemplo de vista com ASP.NET view engine

- Razor *view engine* – nova plataforma introduzia com ASP.NET MVC 3, e que fornece uma sintaxe mais simples e intuitiva, ilustrada com um exemplo na Figura 17.

```
@{
    ViewBag.Title = "Error";
}

<hgroup class="title">
    <h1 class="error">Error.</h1>
    <h2 class="error">An error occurred while processing your request.</h2>
</hgroup>
```

Figura 17 – exemplo de vista com Razor view engine

Em ASP.NET MVC é introduzido um novo conceito, os *HTML Helpers*. Estes artefactos são um conjunto de classes e métodos cujo objetivo é facilitar ao programador a interação com os principais elementos de HTML, um pouco à semelhança dos *form controls* dos Web Forms.

```
<div class="form-group">
    <div class="col-md-offset-2 col-md-10">
        <div class="checkbox">
            @Html.CheckBoxFor(m => m.RememberMe)
            @Html.LabelFor(m => m.RememberMe)
        </div>
    </div>
</div>
```

Figura 18 – exemplo de HTML Helpers

Outro elemento essencial para o correto funcionamento deste modelo é o mecanismo de troca de informação entre os diferentes elementos que compõe o padrão de desenho MVC. Os controladores são os elementos invocados pelo cliente HTTP, e é nesses

controladores que é executada lógica que pode ou não culminar com a devolução de uma vista. Sucede que é essencial que exista troca de informação entre esses dois elementos. A *framework* não se limita a oferecer uma solução para este problema, oferecendo mesmo três diferentes opções:

- ViewData – dicionário de dados do tipo chave-valor que é acessível no controlador e vistas correspondentes. Este artefacto pode ser alimentado com dados na classe controlador (Figura 19), que por sua vez são acessíveis na vista devolvida pela classe (Figura 20).

```
public ActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    return View();
}
```

Figura 19 – exemplo de utilização de ViewData no controlador

```
<div>
    <h1>Message:</h1>
    <h2>@ViewData["Message"]</h2>
</div>
```

Figura 20 – exemplo de utilização de ViewData na vista

- ViewBag – dicionário de dados do tipo chave-valor com sintaxe simplificada (introduzido no MVC 3). À semelhança do ViewData, a informação é preenchida na classe controlador (Figura 21) para ser acedida na vista devolvida pela mesma (Figura 22).

```
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";

    return View();
}
```

Figura 21 – exemplo de utilização de ViewBag no controlador

```
<div>
    <h1>Message:</h1>
    <h2>@ViewBag.Message</h2>
</div>
```

Figura 22 – exemplo de utilização de ViewBag na vista

- View Object Model – definição de um objeto modelo da estrutura de dados de comunicação entre o controlador e a vista. As vistas que usam este modelo denominam-se *strongly typed views*. As classes que definem *strongly typed views* permitem automaticamente definir um conjunto de parâmetros sobre cada propriedade ou elemento, como por exemplo o tipo de dados, o descritivo da *label* que descreve o campo, ou a mensagem de erro caso o campo obrigatório não seja preenchido (Figura 23). A classe permite instanciar objetos que podem ser acessíveis na vista (Figura 24) e controlador correspondentes (Figura 25).

```
public class LoginModel
{
    [Required(ErrorMessage="UserName is required!")]
    [Display(Name="UserName")]
    public string UserName { get; set; }

    [Required(ErrorMessage = "Passowrd is required!")]
    [Display(Name = "Password")]
    [DataType(DataType.Password)]
    public string Password { get; set; }
}
```

Figura 23 – exemplo de View Model

```
<div class="form-group">
    @Html.LabelFor(m => m.UserName, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })
        @Html.ValidationMessageFor(m => m.UserName)
    </div>
</div>
<div class="form-group">
    @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
        @Html.ValidationMessageFor(m => m.Password)
    </div>
</div>
```

Figura 24 – exemplo de acesso ao View Model na vista

```
public ActionResult Login(LoginModel loginInfo)
{
    if (AuthManager.Login(loginInfo.UserName, loginInfo.Password))
    {
        return View("LoginSuccess");
    }
    else
    {
        return View("LoginFailure");
    }
}
```

Figura 25 – exemplo de acesso ao View Model no controlador

3.1.2.3 Modelo

O modelo é o terceiro componente fundamental no padrão de desenho MVC. Existem vários tipos de modelo quando se implementam aplicações web, sendo que os mais relevantes são:

- Data Model – os objetos do Data Model representam classes de interação com dados, normalmente de uma base de dados relacional. Estas classes podem ser, por exemplo, as classes da Entity Framework para interação com a base de dados.
- Business Model – classes que implementam regras de negócio e de processamento de informação. Tipicamente as classes do Business Model interagem com as classes do Data Model para obterem e depositarem informação.
- View Model – classes que modelam a informação que transita entre o controlador e a vista, e a forma como essa informação deve ser apresentada no *browser*.

Qualquer uma destas classes é acessível ao nível do controlador com o objetivo de fazer todo o processamento necessário para cada um dos *action methods* implementado.

As classes do tipo View Model têm a particularidade de serem acessíveis ao nível da vista (como foi descrito no tópico anterior). O transporte da informação com recurso ao View Model é bidirecional, isto é, não só é possível aceder à informação construída no controlador pelo lado da vista, através por exemplo de um acesso à base de dados, como também é possível, a partir do controlador, usar o View Model para obter informação de um formulário (criado por uma vista) que esteja ligado a esse objeto. Este tipo de vistas são denominados Strongly Typed Views.

3.1.3 Estruturação de páginas em ASP.NET MVC

Como seria de esperar, a *framework* ASP.NET MVC implementa soluções que permitem estruturar páginas de forma eficiente, maximizando a reutilização de código e diminuindo a complexidade de componentes, dividindo-os em partes mais pequenas.

A estruturação de elementos visuais depende do *view engine* utilizado (ASP.NET ou Razor).

Caso no projeto ASP.NET MVC se opte pelo *view engine* ASP.NET, será possível tal como em ASP.NET Web Forms definir *master pages* capazes de agrupar elementos repetidos em várias páginas, tais como o cabeçalho, o rodapé ou o menu. Estas *master pages* podem implementar vários níveis de estrutura, através de herança visual.

No entanto o conceito de página e controlos do ASP.NET Web Forms não existe em MVC. São as vistas que irão definir os controlos visuais a serem implementados na *master page*, em cada pedido HTTP. Por sua vez, as *partial views* (vistas parciais) permitem definir elementos visuais que podem ser reutilizados várias vezes dentro de uma vista.

Já com o *view engine* Razor a estruturação de páginas é menos parecida com o modelo Web Forms. A este nível apenas existem vistas, implementadas em ficheiros de extensão cshtml. No entanto as vistas possuem particularidades que lhes permitem implementar comportamentos análogos a conceitos do ASP.NET Web Forms. Por exemplo, em cada vista é possível definir *sections*, que são zonas nas quais o conteúdo pode ser definido por outras vistas. Desta forma poderemos ter vistas com o mesmo efeito das *master pages*. Existe também com o *view engine* Razor o conceito de *partial views*, isto é, vistas que podem ser repetidas diversas vezes no mesmo pedido, implementando o comportamento equivalente aos controlos do ASP.NET Web Forms.

A grande conclusão é que, apesar com nomenclatura e formas diferentes, a forma de estruturar os conteúdos visuais em ASP.NET MVC não é tão diferente de ASP.NET Web Forms como pode parecer à primeira vista.

3.2 Alterações propostas à arquitetura

Com a alteração da tecnologia de suporte a uma determinada aplicação ou sistema, poderá surgir a necessidade ou a oportunidade de alterar a arquitetura existente. No caso concreto deste estudo pretende-se fazer evoluir um portal que já existe, e como tal um eventual redesenho da arquitetura terá de pesar o impacto de *refactoring*, que por vezes é mais pesado do que o tempo da própria implementação. Por consequência, as alterações à arquitetura propostas têm o intuito de a tornar a melhor possível, com um esforço razoável.

Nesse sentido, no que diz respeito à divisão em camadas, propõe-se a seguinte estruturação, ilustrada na Figura 26:

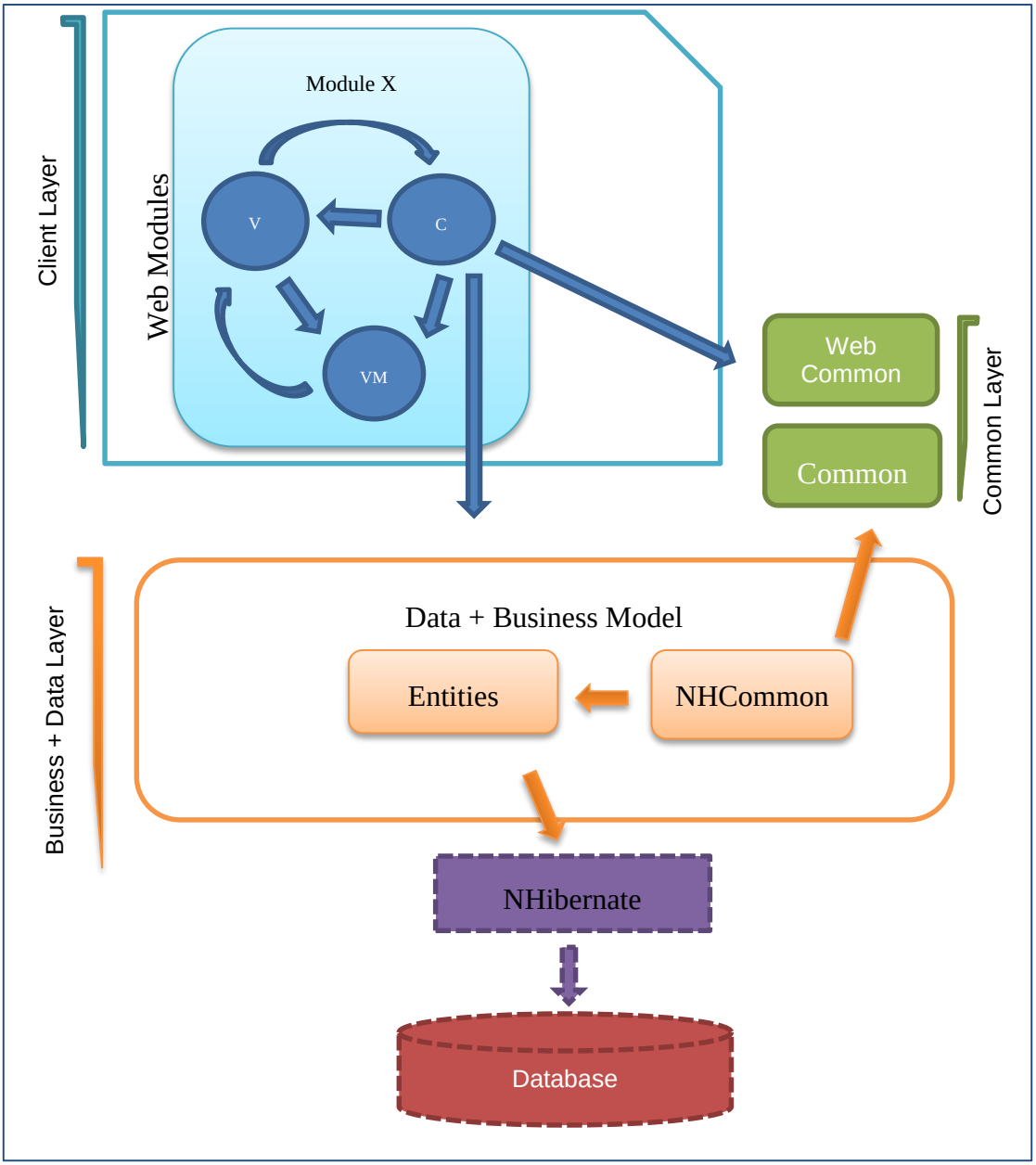


Figura 26 – diagrama de arquitetura proposta

3.2.1 Decomposição em camadas

Os princípios pelos quais foi definida a criação de 3 camadas na arquitetura do EMS mantêm-se, mesmo quando adotando o padrão de desenho MVC, nomeadamente:

- A camada de negócio engloba as classes responsáveis pelos dados e as classes responsáveis pelas regras de negócio. É essencial este componente ser independente de qualquer modelo de apresentação para, no limite, todo o código ser reutilizável na implementação de uma aplicação diferente, como por

exemplo uma aplicação Windows Forms. Na camada de negócio existem classes relativas às entidades e à interface com a base de dados, correspondendo aos elementos de Data Model e Business Model do padrão MVC. Em termos gerais não será necessário efetuar alterações nas classes existentes nesta camada de negócio.

- A camada comum contém um conjunto de classes conhecidas por todas as camadas da aplicação e que definem os tipos de dados conhecidos em comum pelas camadas cliente e de negócio. É pouco relevante ao nível do *refactoring* necessário para adotar o padrão MVC, e não deverá ser alvo de alterações.
- A camada cliente será aquela que irá sofrer maior volume de alterações. É a camada responsável pela interface entre o utilizador e o negócio, fornecendo os formulários ao utilizador e tirando partido das classes de negócio para fazer a ligação com a base de dados. Dentro desta camada cada módulo será migrado para MVC, e portanto toda a lógica que atualmente existe nos ficheiros de *master pages*, páginas e controlos será repartida por vários ficheiros, melhorando as fronteiras de responsabilidade entre blocos de código:
    - View – cada módulo terá uma ou mais vistas para implementar o correspondente às atuais master pages, páginas ou controlos. Nestes ficheiros de código apenas deverá existir lógica com a responsabilidade de definir os formulários apresentados ao utilizador.
    - Controller – cada módulo deverá ser constituído pelas classes controlador necessárias e respetivos *action methods*, de forma a ser a interface de entrada de todos os pedidos dos utilizadores no portal. Cada controlador deverá implementar a lógica própria do portal, sendo que a implementação das regras próprias do negócio e dos dados deve ser direcionada para os Business Model e Data Model.
    - View Model – cada módulo poderá ser dotado de uma ou mais classes representativas da modelo das vistas apresentadas, constituindo interfaces de comunicação entre a vista e o controlador.

3.2.2 Decomposição de camada cliente em artefactos ASP.NET MVC

Tal como na arquitetura anterior, o portal EMS será dividido em módulos que agruparão funcionalidades e formulários de acordo com a sua afinidade em termos de negócio. Esses módulos, por sua vez, são divididos em diversos elementos de código, de forma a promover a reutilização, diminuir a repetição de código e a complexidade.

Para implementar cada módulo da forma descrita, irá tirar-se partido das capacidades da *framework* ASP.NET MVC, com o ASP.NET *view engine*. É importante referir este

pormenor neste ponto, uma vez que o *view engine* delineia a forma como as vistas estão estruturadas.

Ir-se-á portanto restruturar o código das vistas, sendo que em cada módulo existirão *master pages*, *views* (vistas), (onde em Web Forms existiam as páginas) e *partial views* (vistas parciais), (onde em Web Forms existiam os controlos). Cada um dos elementos dos vários tipos deverá ter uma classe controlador para implementação do *code behind*.

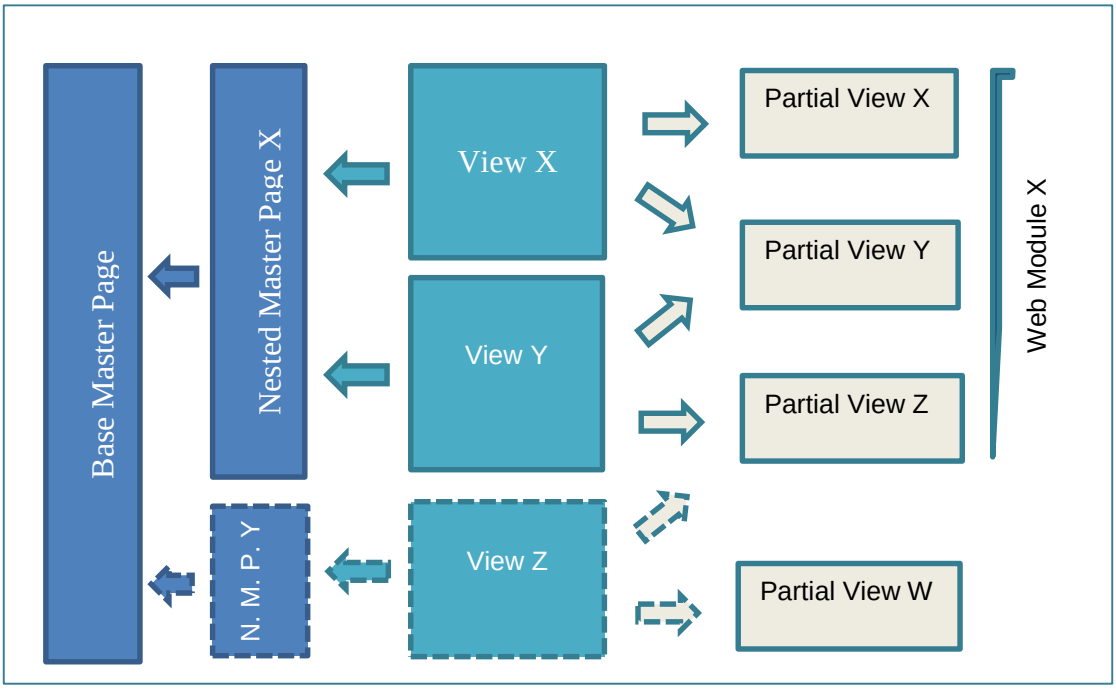


Figura 27 – estruturação de módulos com nova arquitetura

Não estaria a fazer um estudo de uma evolução para a tecnologia ASP.NET MVC se não existissem fortes indícios de que existem oportunidades inerentes, que constituem mais-valias importantes para o produto e respetivo processo de desenvolvimento.

3.3.1 Desempenho

A abordagem MVC é, comparativamente ao modelo Web Forms, bastante mais próxima das tecnologias de suporte, e isso pode ser uma interessante vantagem em termos de desempenho.

Um maior controlo sobre a forma como o HTML, os scripts e os pedidos HTTP são executados dá ao programador maior controlo sobre todas as ações, o que lhe permite

perceber, por exemplo, quando uma determinada opção pode ou não gerar um pedido HTTP excessivamente grande, quando o *browser* está a puxar um script desnecessário, ou quando deve ser feito num pedido do tipo GET em vez de POST (tipicamente mais pesado).

Todo este controlo dá ao programador a noção exata do impacto que determinado bloco de código ou opção técnica tem no desempenho global da aplicação. Em ASP.NET Web Forms a abordagem é simplificada para o programador, sobretudo em relação a programadores mais inexperientes, que não têm a noção do impacto que determinada opção pode ter no desempenho da aplicação.

É portanto provável que uma migração para a *framework* MVC possa ser uma mais-valia importante, na medida em que tornará uma aplicação mais rápida.

3.3.2 Separação de conceitos

MVC significa Modelo, Vista e Controlador, e são esses os conceitos ou as fronteiras de abstração que teremos ao implementar código usando esta *framework*.

Essa flexibilidade permitirá uma maior e melhor estruturação do código, com potencial impacto na produtividade do programador e na manutenção do produto. Uma clara distinção entre vista, modelo e controlador trará uma mais rápida e eficiente abordagem da equipa de manutenção em caso de necessidades evolutivas ou corretivas.

Para além disso, e uma vez que o produto EMS é mantido neste momento para diversos clientes, a separação de conceitos pode ser vantajosa para implementar comportamentos díspares entre eles. Por exemplo, num determinado módulo, podemos ter comportamentos particulares implementados por um controlador único, mantendo válido todo o código da vista e modelo respetivos. Ou podemos ainda ter uma vista diferente para um diferente cliente, sem necessidades maiores de *refactoring* nos restantes componentes de código. Os conceitos de herança e polimorfismo são mais facilmente aplicáveis em MVC.

3.3.3 Testabilidade

A separação da lógica de controlo da lógica de apresentação vem dotar a *framework* MVC de uma maior testabilidade, principalmente por separar de uma forma mais clara a lógica de controlo da lógica responsável pela apresentação (Etheredge, 2009).

Será uma oportunidade para implementar mais e melhores testes unitários, o que, quando automatizados, iriam garantir com certeza outro tipo de garantias de qualidade, uma vez que o número de defeitos causados com *refactoring* de código iria diminuir, e o esforço necessário para testar e os corrigir iria diminuir também.

4 Prova de conceito

O EMS é neste momento um projeto maduro, instalado em vários clientes, com atividades de manutenção corretiva e evolutiva a decorrer e um *roadmap* [27]definido para o futuro do produto. Nesta fase do seu ciclo de vida, interessa não só perceber se de facto existem vantagens claras em mudar a *framework* base do código do projeto de ASP.NET Web Forms para ASP.NET MVC, mas também até que ponto esse processo é viável e compatível com as tarefas a realizar no dia-a-dia da equipa de projeto do EMS.

Neste trabalho a prova de conceito consiste em estudar e documentar um processo para tornar a solução do EMS capaz de suportar a evolução tecnológica de ASP.NET Web Forms para ASP.NET MVC. O processo utilizado terá de ser compatível com o ciclo de vida do produto, e o menos intrusivo possível nas atividades realizadas pela equipa de projeto, e que permita, num período de transição, a coexistência entre as duas tecnologias. Dessa forma, e melhor do que fazer uma mudança drástica na arquitetura da solução, com riscos acrescidos para o projeto e um grande impacto no processo de desenvolvimento, estudou-se a viabilidade de montar uma solução capaz de fazer essa mudança de forma gradual, durante o ciclo de vida do produto. Essa solução irá permitir que novos módulos ou implementações possam ser já implementados em ASP.NET MVC, coexistindo com os módulos atualmente implementados em ASP.NET Web Forms de forma transparente para os utilizadores finais (Esposito 2014).

Após a implementação desse processo, foi estudada a viabilidade de migração de um módulo existente atualmente no EMS, implementado em ASP.NET Web Forms, para ASP.NET MVC. Com a solução montada, que permite a coexistência de ambas as tecnologias, uma eventual migração dos componentes ASP.NET Web Forms poderá ser faseada no tempo, de acordo com a disponibilidade da equipa e as prioridades do projeto, até se chegar a um ponto em que se poderá de todo remover essa tecnologia da solução, passando a usar-se somente ASP.NET MVC. No entanto interessa perceber, por um lado, a complexidade da operação, por outro lado quais os reais benefícios que representa essa migração, informações relevantes para decidir como será a evolução futura do produto EMS.

[27] O *roadmap* num projeto de software é um plano que combina objetivos de curto e longo prazo com as soluções tecnológicas que possibilitam atingir esses objetivos.

4.1 Processo de adequação da solução

Esta secção descreve os principais passos que foram executados durante o processo de adequação da solução do projeto à nova *framework*.

4.1.1 Setup do ambiente

Este projeto foi executado paralelamente ao decurso das atividades normais e recorrentes do EMS enquanto produto.

Como tal, o primeiro passo para a montagem do ambiente de desenvolvimento deste trabalho foi a criação de um ramo de desenvolvimento a partir da última versão do produto, a 2.1, no servidor de ficheiros SVN, indispensável para que os trabalhos realizados na configuração da solução à nova realidade não condicionasse nem fosse condicionada pelas atividades normais de manutenção e evolução do produto.

Como ilustrado no diagrama da Figura 28, sobre a *tag* da versão 2.1 do produto foi criado um *branch*, que permitiu a realização dos trabalhos que envolvem a solução de código do projeto de forma paralela e independente das restantes atividades da equipa de projeto.

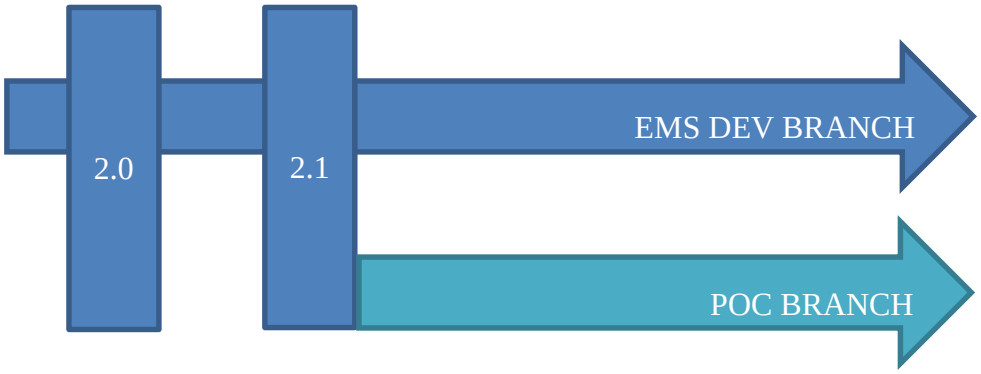


Figura 28 – estratégia de Configuration Management

Para além dessa separação do código foi também criado um clone do *schema* de base de dados, com base nos scripts de criação de base de dados marcados com a *tag* da versão 2.1. do produto.

Desta forma este projeto pôde ser totalmente independente do normal desenvolvimento e manutenção realizados pela equipa do EMS, quer ao nível do código, quer ao nível dos dados de teste.

Para além do código e da base de dados de teste, o ambiente de desenvolvimento é composto por uma máquina virtual com as ferramentas necessárias, nomeadamente:

- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2013
- IIS 7
- Devexpress 2010 (componentes gráficos)

4.1.2 IDE adaptado às necessidades

O processo de evolução tecnológica adotado trouxe algumas necessidades especiais relativamente ao IDE, neste caso ao Visual Studio (no arranque destes trabalhos estava em uso o Visual Studio 2010, que foi evoluído para o 2013, conforme será descrito de seguida).

O problema prende-se pelo facto de pretendermos passar a incluir elementos de código ASP.NET MVC num projeto Visual Studio do tipo ASP.NET Web Forms. Todas as opções que o Visual Studio nos oferece estão focadas no modelo de desenvolvimento Web Forms. Por exemplo, quando pretendemos adicionar um novo item ao projeto, as opções por omissão não incluem elementos como *View* ou *Controller*, como é visível na ilustração da Figura 29.

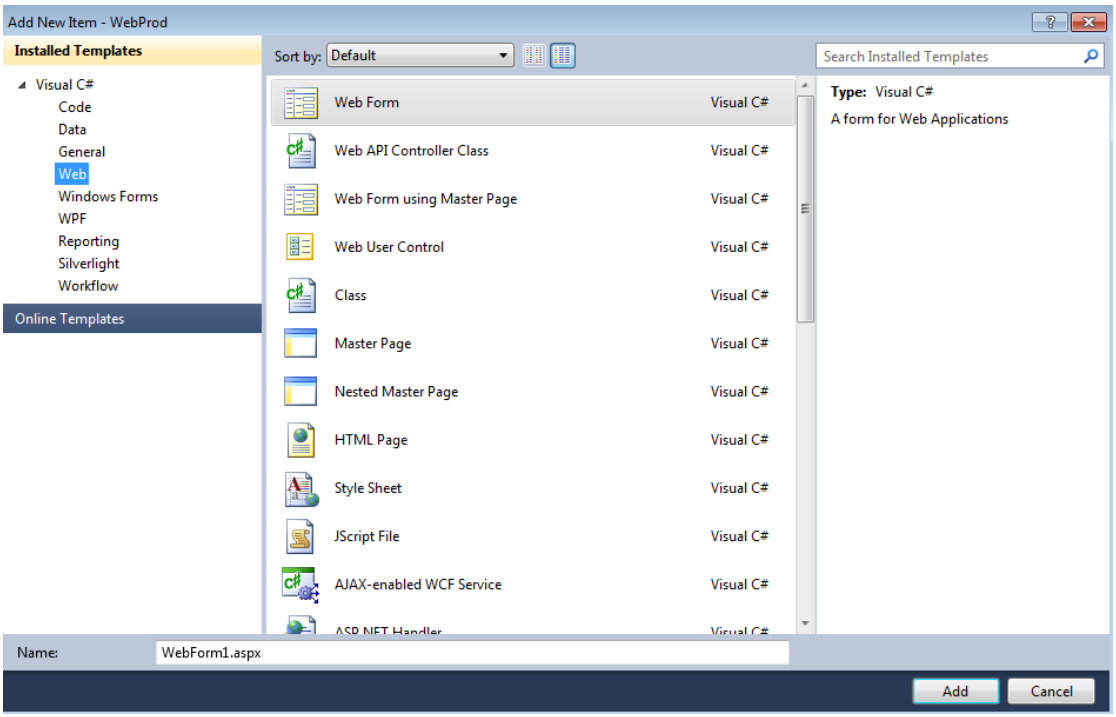


Figura 29 – opções disponibilizadas pelo Visual Studio 2010 para adicionar novos items ao projecto ASP.NET Web Forms

Investigando sobre o problema na comunidade de programadores na Internet, confirmei a minha convicção de que elementos de ASP.NET Web Forms e ASP.NET MVC podem coexistir no mesmo projeto, pois partilham uma base comum. No entanto, em alguns exemplos a solução utilizada era a construção de raiz dos elementos de um projeto MVC, isto é, criar ficheiros de código "limpos" que seriam customizados manualmente para se tornarem vistas ou controladores, sem tirar partido de qualquer *template* do Visual Studio. Esta solução, embora viável em último recurso, tornaria o processo bastante penoso e sujeito a erros.

Outra solução utilizada consiste na criação de um novo projeto ASP.NET MVC e adição manual de todos os elementos do projeto ASP.NET Web Forms existente. Esta solução, também ela demorada e penosa, tem um problema com o qual não poderíamos lidar no projeto EMS: o Visual Studio 2010 passaria a suportar ASP.NET MVC e não ASP.NET Web Forms, algo que iria contra a nossa estratégia de encontrar uma solução "híbrida", para que a evolução para MVC seja o menos intrusiva possível no decorrer do ciclo de vida do produto.

No entanto a Microsoft esteve atenta a esta limitação e, como investiguei, a versão 2013 do Microsoft Visual Studio já permite a coexistência de ambas as *frameworks* no mesmo projeto. Resolvi portanto proceder à migração para a nova versão da ferramenta.

Após passar pelo processo de instalação da nova versão do IDE, com todas os *service packs* e *updates* disponíveis, verificou-se que a nova aplicação, embora torne possível adicionar vistas e controladores ao projeto em migração, apenas possibilita selecionar o *view engine* Razor, que não é o que pretendemos usar neste projeto! Como é visível na Figura 30, os *templates* disponíveis para elementos MVC apenas incluem o *view engine* Razor.

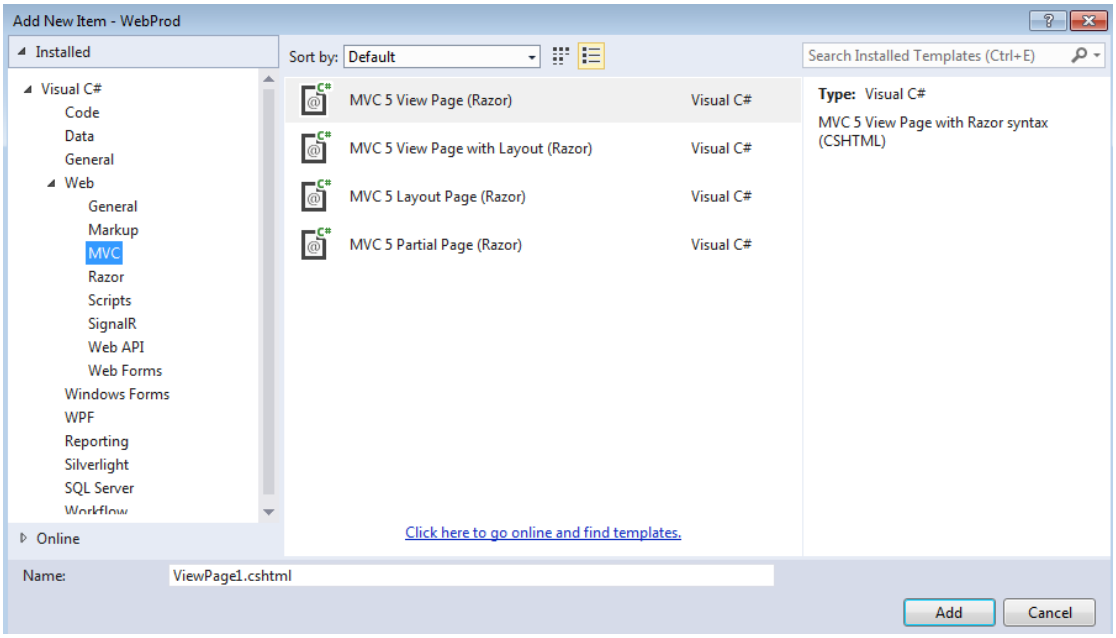


Figura 30 – opções disponibilizadas pelo Visual Studio 2013 para adicionar items MVC a um projeto ASP.NET Web Forms

A utilização do *view engine* Razor, que possui uma sintaxe mais simples e para o qual a última versão do IDE Visual Studio 2013 direciona os programadores, seria mais intrusiva neste processo de migração, como já referimos, uma vez que a sua coexistência com os elementos ASP.NET Web Forms seria mais complicada, e como tal a análise continuou no sentido de encontrar a solução perfeito (ou o mais próximo disso).

E conseguiu-se chegar a uma solução bastante próxima da ideal, seguindo o seguinte processo:

- Criação de um novo projeto *dummy* ASP.NET MVC 4.0 com *view engine* ASP.NET, com Visual Studio 2013, como ilustrado na Figura 31.

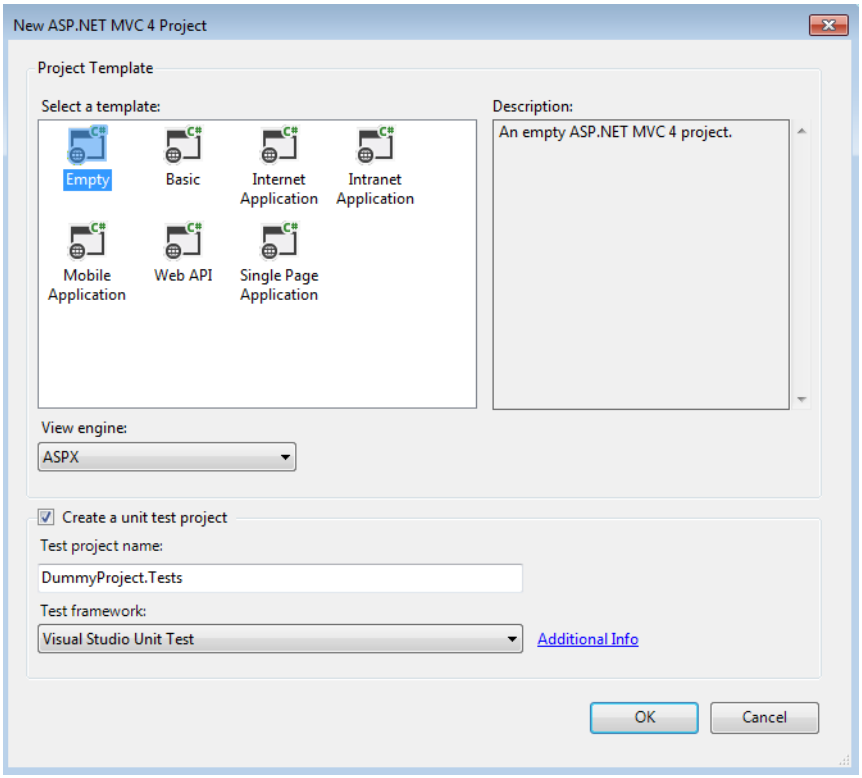


Figura 31- criar projeto ASP.NET MVC com Visual Studio 2013

- Validação das opções deste tipo de projeto, ao nível do IDE. Como é visível na Figura 32, usando este *template* de projeto o Visual Studio passa a disponibilizar *templates* para classes de código MVC com *view engine* ASP.NET.

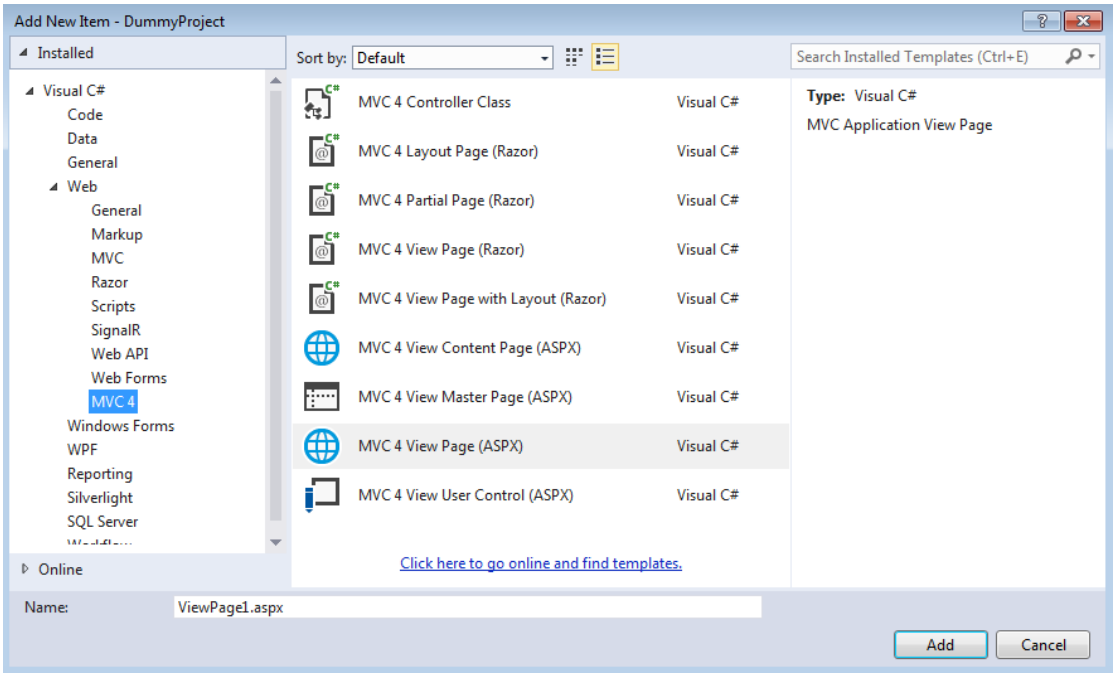


Figura 32- opções disponíveis para adicionar novo item no novo projeto

- Fechar o Visual Studio e abrir o ficheiro do projeto recentemente criado com o Notepad, e copiar as *hash* relativas ao tipo de *template* de projeto, que corresponde à área a sombreado da Figura 33. Este campo possui um conjunto de *hash codes* que correspondem a *templates* de funcionalidade que o Visual Studio proporciona nos projetos.

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="12.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props" Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>
    </ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{7EB25A3C-008C-4A01-8D09-06D5E319EE64}</ProjectGuid>
    <ProjectTypeGuids>{E3E379DF-F4C6-4180-9B81-6769533ABE47};{349c5851-65df-11da-9384-00065b846f21};{fae04ec0-301f-11d3-bf4b-00c04f79efbc}</ProjectTypeGuids>
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>DummyProject</RootNamespace>
    <AssemblyName>DummyProject</AssemblyName>
    <TargetFrameworkVersion>v4.0</TargetFrameworkVersion>
    <MvcBuildViews>false</MvcBuildViews>
    <UseIISExpress>true</UseIISExpress>
    <IISExpressSSLPort />
    <IISExpressAnonymousAuthentication />
    <IISExpressWindowsAuthentication />
    <IISExpressUseClassicPipelineMode />
  </PropertyGroup>
```

Figura 33- opção project type no novo projeto

- Abrir o ficheiro do projeto que pretendemos migrar também com Notepad e completar com as *hash* inexistentes. Esta técnica irá fazer com que o Visual Studio passe a disponibilizar funcionalidades relativas aos projetos MVC no projeto existente.
- Confirmar que o tipo de opções disponíveis para adicionar ao projeto é o mesmo do projeto *dummy* criado, como por exemplo as opções ilustradas na Figura 32.

Confirmou-se que esta solução resolveu o problema detetado. Converteu-se manualmente o *template* do projeto Visual Studio para um projeto ASP.NET MVC, com *view engine* ASP.NET. Este processo é o equivalente à solução de criar um novo projeto e importar todos os ficheiros, mas revelou-se mais rápido e eficaz, e menos sujeito a erros.

A partir deste momento a solução do Portal no Visual Studio permite adicionar e operar não só apenas em elementos ASP.NET Web Forms, mas também com elementos ASP.NET MVC, de forma completamente natural.

4.1.3 Adaptação da solução aos elementos MVC

Uma vez configurado o Visual Studio para facilitar e permitir a inclusão de elementos MVC na solução, é necessário proceder à adaptação da solução, como por exemplo criação de uma estrutura de pastas para classes modelo, vista e controlador, ou a adição das referências a librarias da *framework* .NET necessárias para criação dos tipos de dados MVC.

O primeiro passo aqui foi, mais uma vez, olhar para um projeto ASP.NET MVC criado no Visual Studio, para servir de referência. A Figura 34 ilustra a estrutura base de pastas e ficheiros.

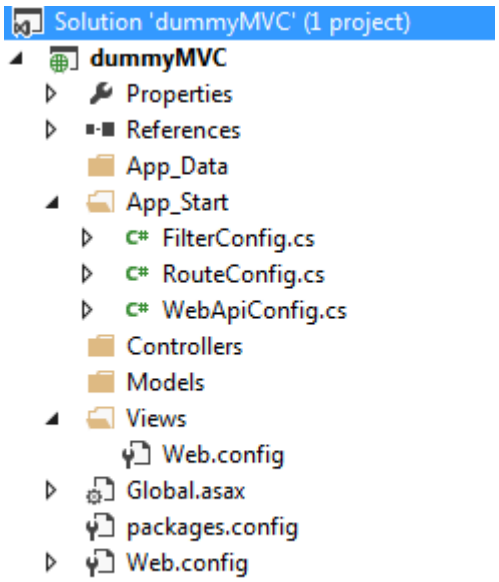


Figura 34 – estrutura de pastas e ficheiros base num projeto ASP.NET MVC

Relativamente a pastas e ficheiros, a estratégia passou por criar na nossa solução as pastas inexistentes. Relativamente aos ficheiros, foram copiados os ficheiros que não existiam na nossa solução. A Figura 35 ilustra o resultado final dessa adaptação.

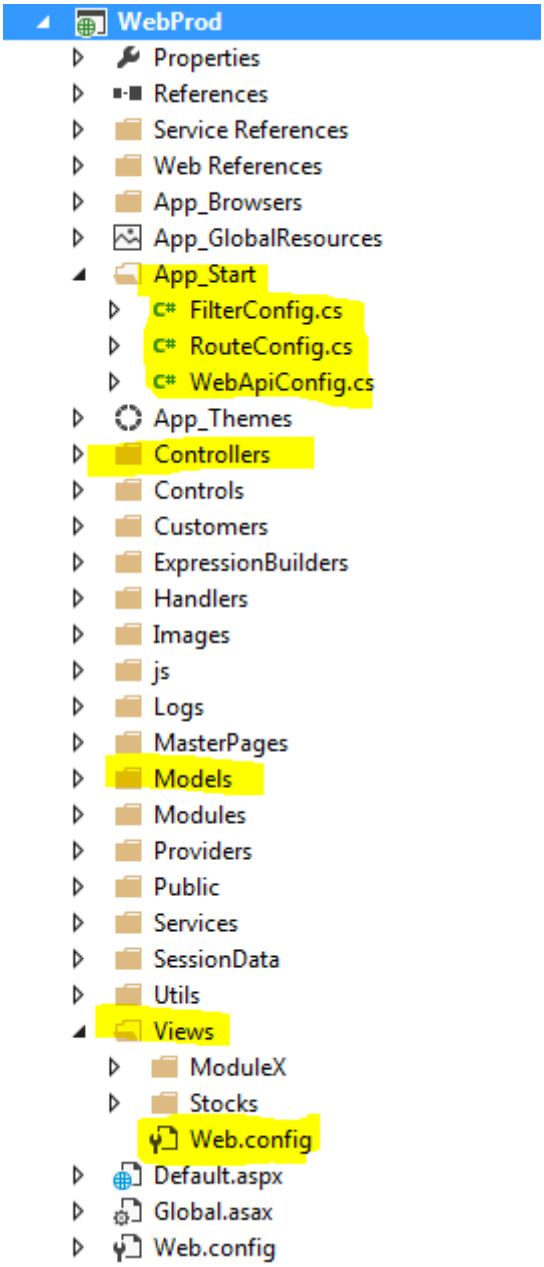


Figura 35 – estrutura da solução EMS adaptada ao MVC

Em ficheiros existentes quer na solução MVC quer na solução EMS existente, como por exemplo o Global.asax, ficheiro onde é colocado código relativo aos eventos da aplicação web, foi analisado o seu conteúdo (Figura 36) e importado o código para o correspondente na nossa solução (Figura 37) (Appel, 2013).

```
namespace dummyMVC
{
    // Note: For instructions on enabling IIS6 or IIS7 classic mode,
    // visit http://go.microsoft.com/?LinkId=9394801
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();

            WebApiConfig.Register(GlobalConfiguration.Configuration);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

Figura 36- conteúdo do ficheiro Global.asax do projecto MVC

```
        protected void Application_Start(Object sender, EventArgs e)
        {
#if DEBUG
            log4net.Config.XmlConfigurator.Configure();
#endif
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            this.SetupServiceLocator();
            this.SetupRoutes();

            AreaRegistration.RegisterAllAreas();

            WebApiConfig.Register(GlobalConfiguration.Configuration);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);

#if !DEBUG
            //this.Preload();
#endif

            // Handle exceptions thrown inside DevExpress callbacks.

            DevExpress.Web.ASPxClasses.ASPxWebControl.CallbackError += new EventHandler(Application_Error);
        }
```

Figura 37 – adaptação do ficheiro Global.asax no projeto EMS

Outra intervenção essencial para a solução EMS passou por garantir que todas as referências da solução MVC (Figura 38) eram incluídas na solução do EMS. Só desta forma o código MVC implementado poderia ser executado.

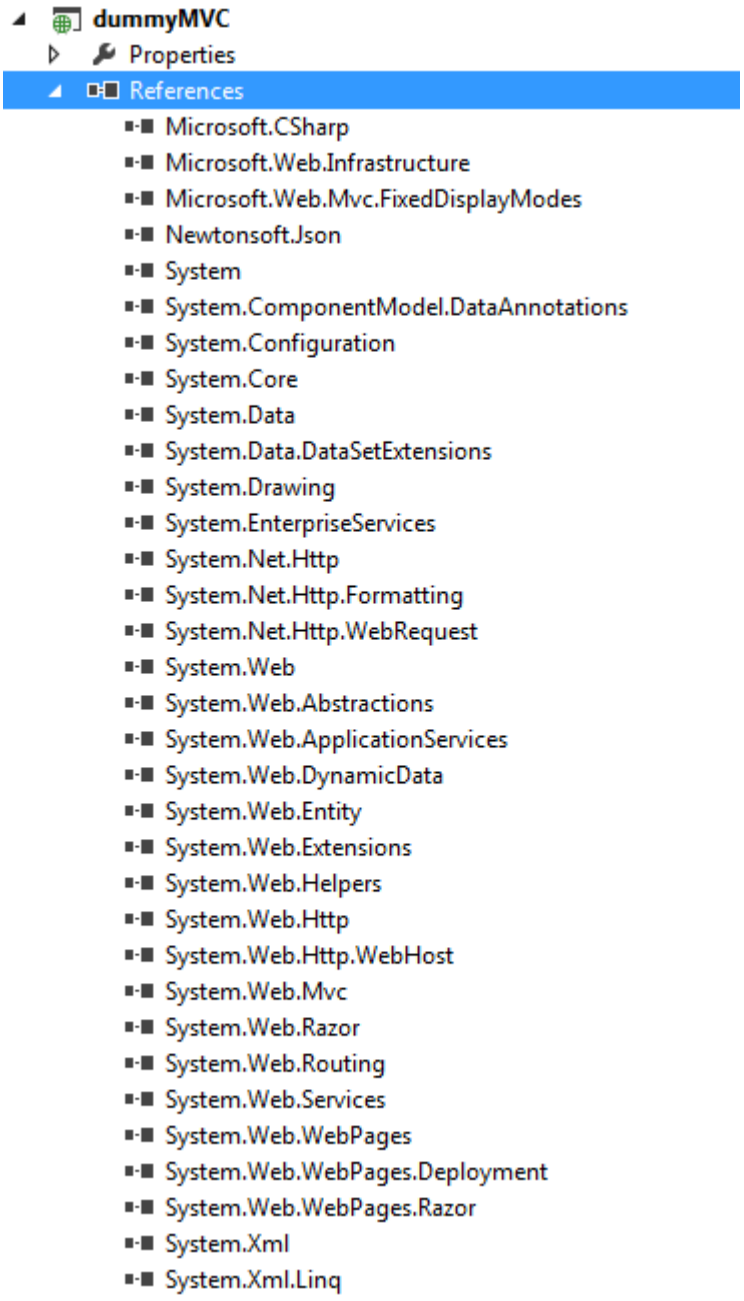


Figura 38 – referências por omissão num projeto ASP.NET MVC

O Visual Studio 2013 possibilita fazer *copy paste* de referências de projeto, pelo que bastou selecionar todas as entradas no projeto MVC, copiar, e colar na zona correspondente da solução do EMS.

4.1.4 Implementação de um novo módulo MVC

A realização deste trabalho perspetiva que a solução de projeto implementada seja capaz de, no decorrer do ciclo de vida do produto EMS, implementar um novo módulo

utilizando a *framework* ASP.NET MVC, capaz de coexistir com os componentes ASP.NET Web Forms de forma transparente para o utilizador.

Uma vez que o *framework* ASP.NET MVC partilha uma base comum, a integração das duas tecnologias é viável, com ilustrado na Figura 39.

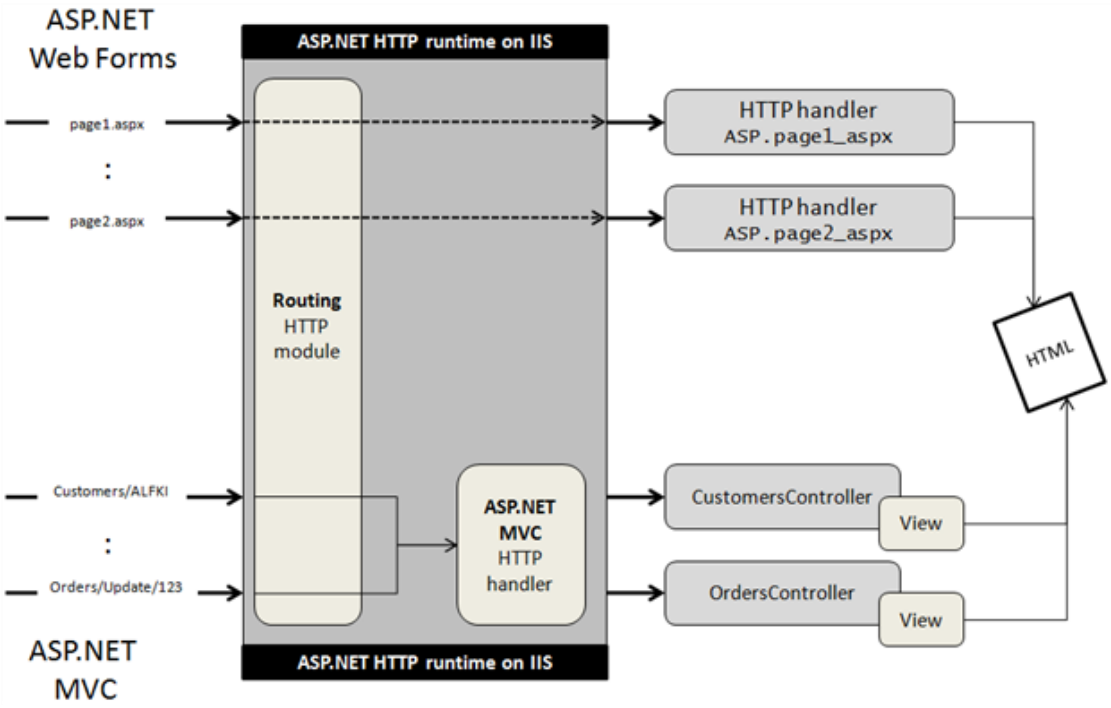


Figura 39 –elementos Web Forms e MVC em simultâneo no IIS (Esposito 2014)

Sendo o output de qualquer página web HTML, será pouco percetível para o utilizador final qual a tecnologia que sustenta o URL acedido, e ainda uma vez que recursos estáticos com imagens, scripts ou CSS podem ser partilhados pelas duas tecnologias. O URL é, no entanto, uma das principais diferenças percetíveis para o utilizador.

Num cenário real, em que queiramos, por exemplo, começar a implementar os novos módulos do EMS em MVC, devemos criar uma estrutura de *master pages* com os elementos que queiramos usamos como *template* entre os diversos módulos. No caso do EMS possivelmente seria necessário criar uma *master page* com o padrão MVC equivalente à *master page* que existe atualmente. Obviamente que esse seria um processo algo demorado, mas apenas teria de ser executado uma vez. A partir dessa altura os novos módulos, e mesmo módulos antigos que eventualmente se reescreverem, podem usar essa *master page*.

Uma vez implementada a *master page* com a estrutura base da página, cada módulo poderá ser implementado utilizando os componentes base do padrão MVC, nomeadamente as classes modelo, controlador e vista (Figura 34).

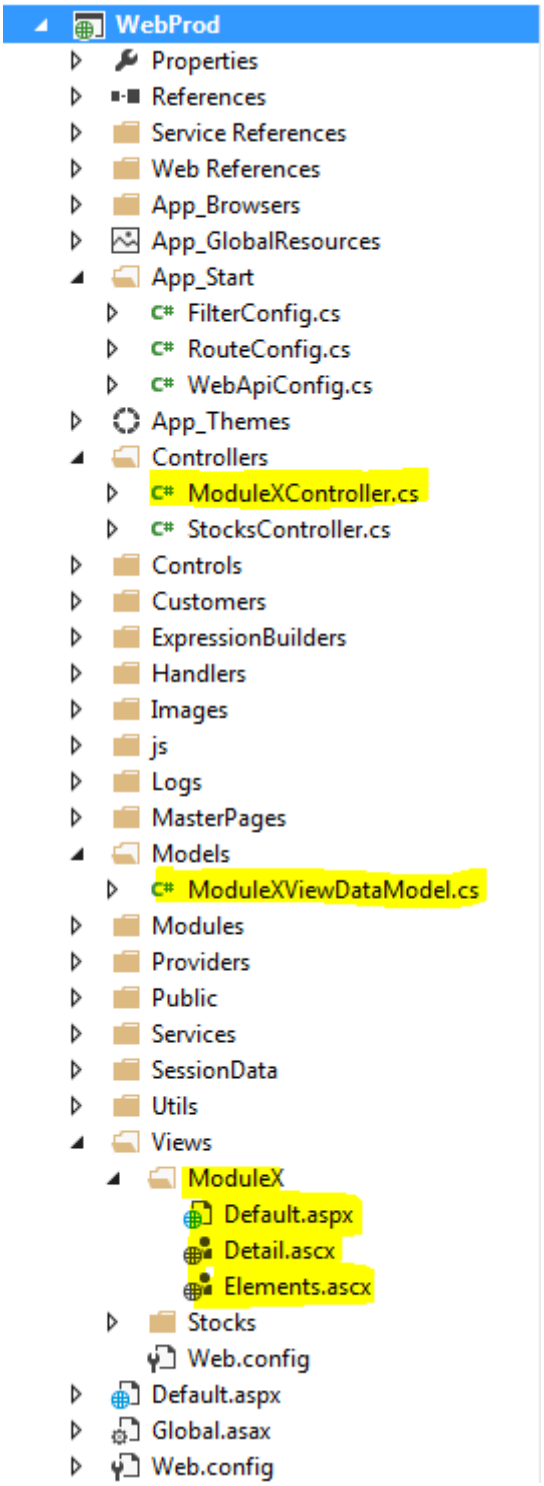


Figura 40 – estrutura base para módulo MVC

Para tornar as funcionalidades implementadas no módulo acessíveis é necessário registar cada a rota relativa a cada URL.

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "ModuleXDefault",
            url: "modules/{controller}/{action}",
            defaults: new { controller = "ModuleX", action = "Index" }
        );
    }
}
```

Figura 41 – rotas para URL do módulo MVC

4.1.5 Migração de um módulo ASP.NET Web Forms existente para ASP.NET MVC

4.1.5.1 Identificação de um módulo de referência

Uma vez criada uma estrutura na solução de projeto que torne possível a coexistência das duas plataformas ASP.NET Web Forms e ASP.NET MVC, será feita uma análise de viabilidade de adequação dos módulos do projeto à nova *framework*. Interessa identificar um módulo ilustrativo e com o maior número de elementos em comum com outros módulos do portal, tornando a análise o mais abrangente possível.

Para este trabalho, foi selecionado o módulo de stocks, devido ao seu carácter genérico comparativamente à maioria dos módulos do sistema, e à dimensão adequada de código a migrar. O ecrã principal do módulo é ilustrado na **Error! Reference source not found.**.

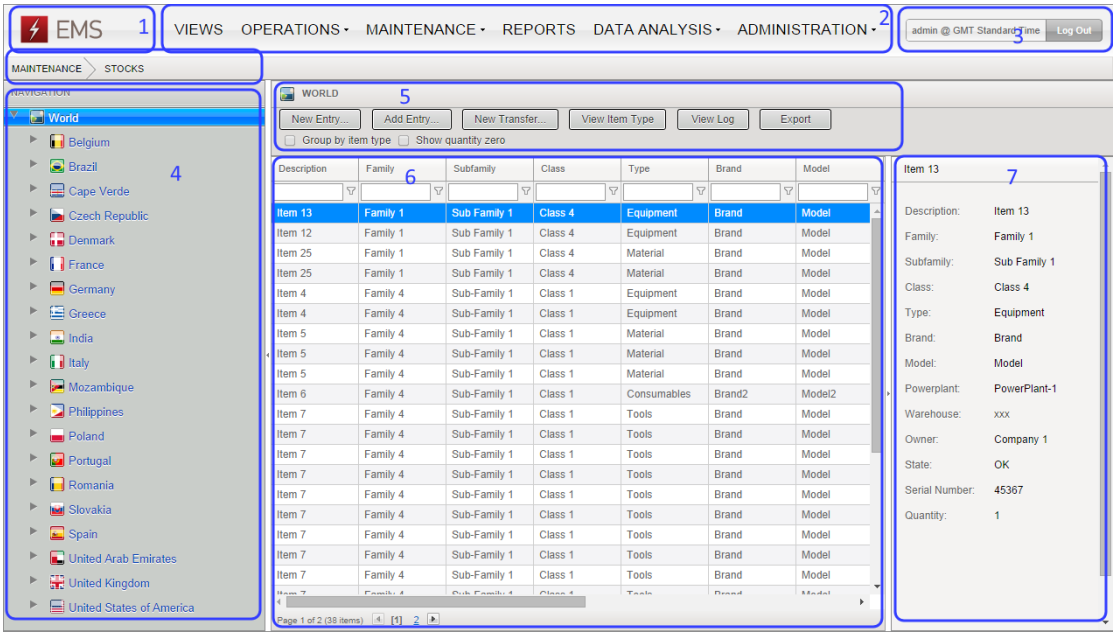


Figura 42 – diferentes zonas de um módulo do EMS

O módulo de stocks, à semelhança da grande maioria dos módulos do EMS, é composto pelas seguintes zonas:

- 1 – Logotipo do produto. Zona comum a praticamente todas as páginas do projeto, e que está implementada ao nível da *master page*.
- 2 – Menu principal. Neste componente está implementada a navegação para as páginas do portal, de acordo com o perfil do utilizador autenticado. Está visível em praticamente todas as páginas do portal e por esse motivo está implementado ao nível da master page.
- 3 – Menu de estado. Apresenta o login do utilizador autenticado e a *timezone* configurada. Apresenta ainda um botão para fazer *logout*. Visível em todas as páginas e implementado na *master page*.
- 4 – Navegação hierárquica. Permite navegar hierarquicamente nas plantas configuradas. Os dados visíveis na grelha e que podem ser consultados e modificados são filtrados, mediante o nó selecionado na árvore. Este componente encontra-se presente em pouco mais de metade das páginas do portal, e como tal foi implementado numa *nested master page*.
- 5 – Painel de ações. Permite ao utilizador despoletar ações sobre os dados presentes ou selecionados na grelha. Muitas das ações existentes, como por exemplo adicionar dados à grelha ou modificar um registo selecionado, são implementadas em *popups* próprios, que são mostrados quando a funcionalidade é acionada. Cada painel de ações depende do módulo selecionado, e como tal o painel de ações é implementado em cada página.
- 6 – Grelha de apresentação dos dados. De acordo com o nó selecionado na árvore e os critérios de filtragem e agrupamento definidos no painel de ações, os dados da base de dados são apresentados na grelha na forma tabelar. A grelha

de dados depende do módulo selecionado, e como tal o painel de ações é implementado em cada página.

- 7 – Painel de detalhes de registo. Este painel apresenta alguns campos de detalhe de informação relacionados com o registo selecionado na grelha. Apesar dos valores apresentados dependerem de cada módulo, este componente encontra-se implementado na *master page*, uma vez que é configurável por parametrização.

Para além dos componentes da página base apresentados, existem ainda os seguintes *popups*:

- Popup new entry (ilustrado na Figura 43): permite inserir novos registos de stocks no sistema. Esses stocks poderão ou não ficar associados a uma Power Plant.

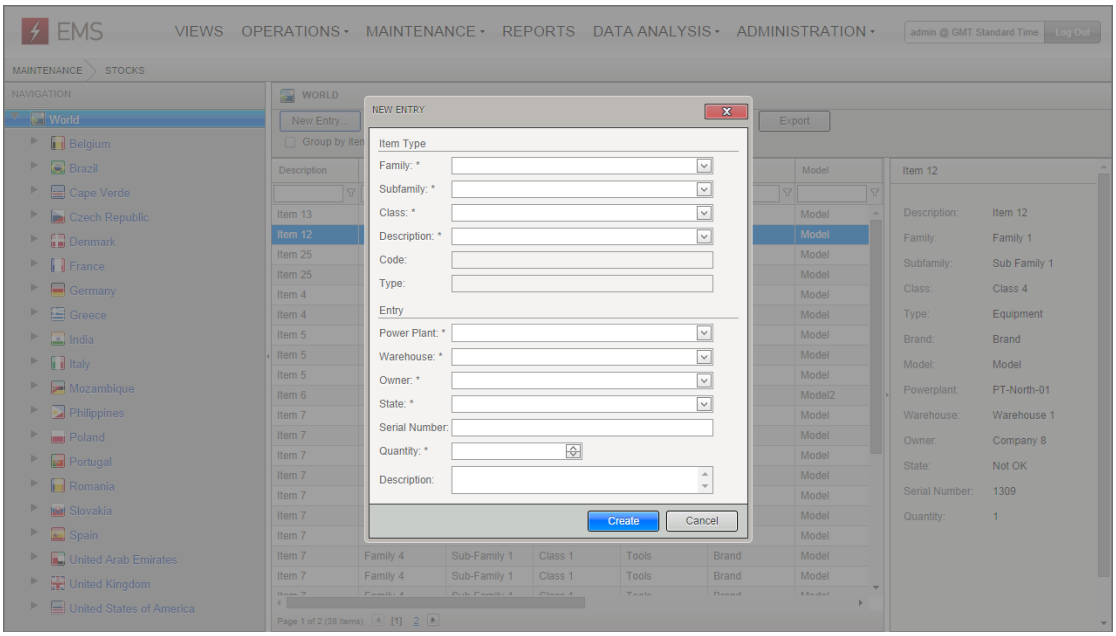


Figura 43 – popup new entry

- Popup add entry (ilustrado na Figura 44**Error! Reference source not found.**): permite inserir registos de stocks no sistema com base no registo selecionado, evitando o preenchimento de todos os campos, alterando apenas os necessários.

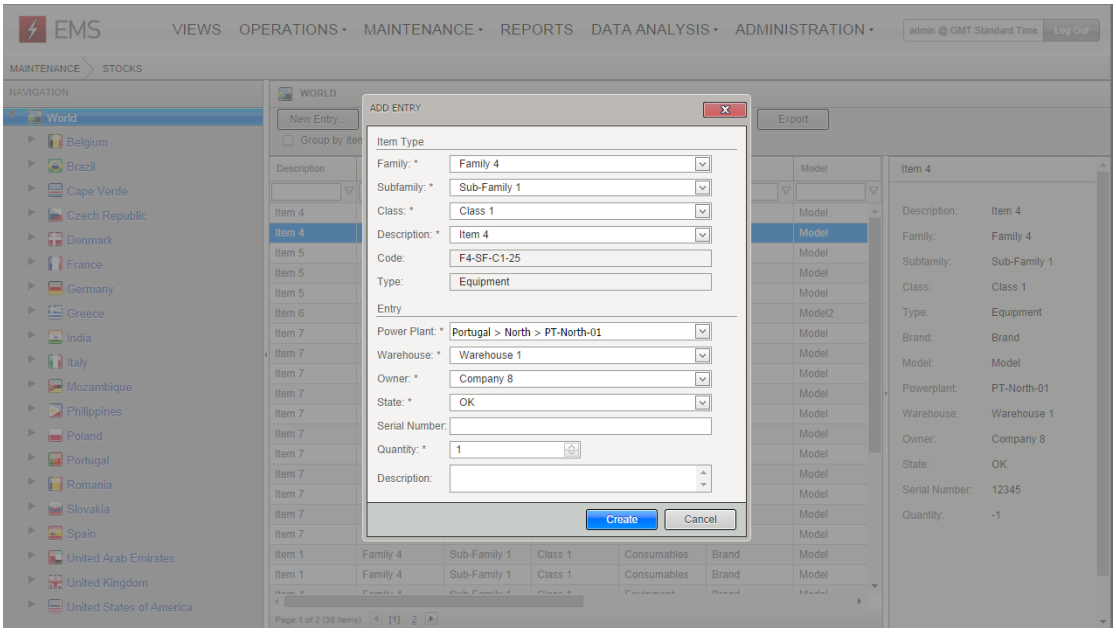


Figura 44 – popup add entry

- Popup new transfer (ilustrado na Figura 45): funcionalidade que permite movimentar existências entre os stocks de diferentes Power Plants.

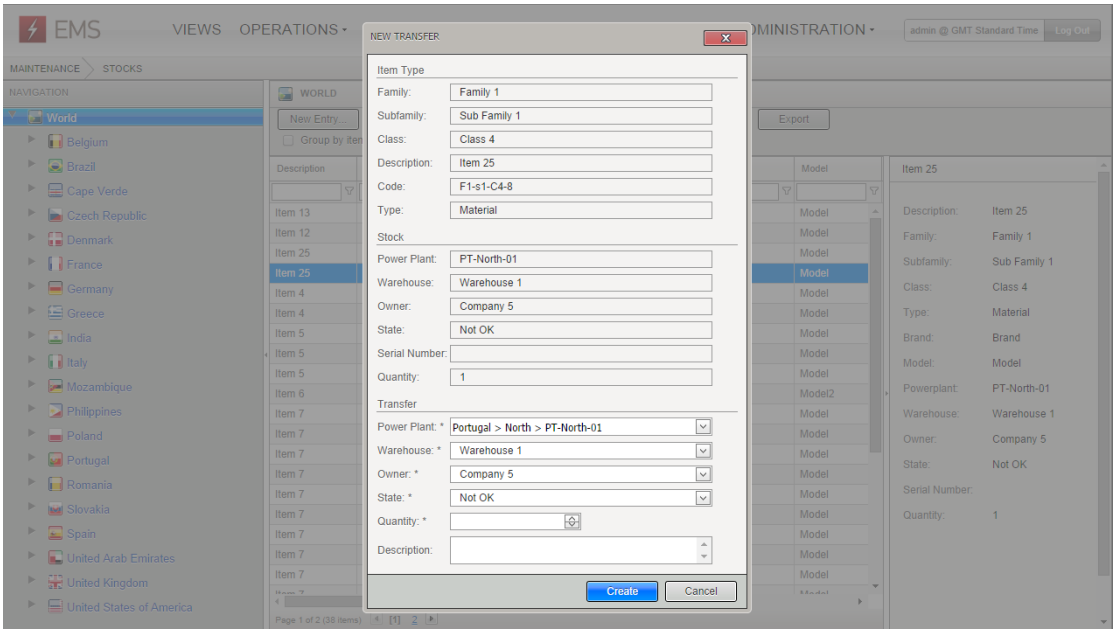


Figura 45 – popup new transfer

- Popup view item type (ilustrado na Figura 46): permite visualizar dados agrupados por item type, para o registo selecionado.

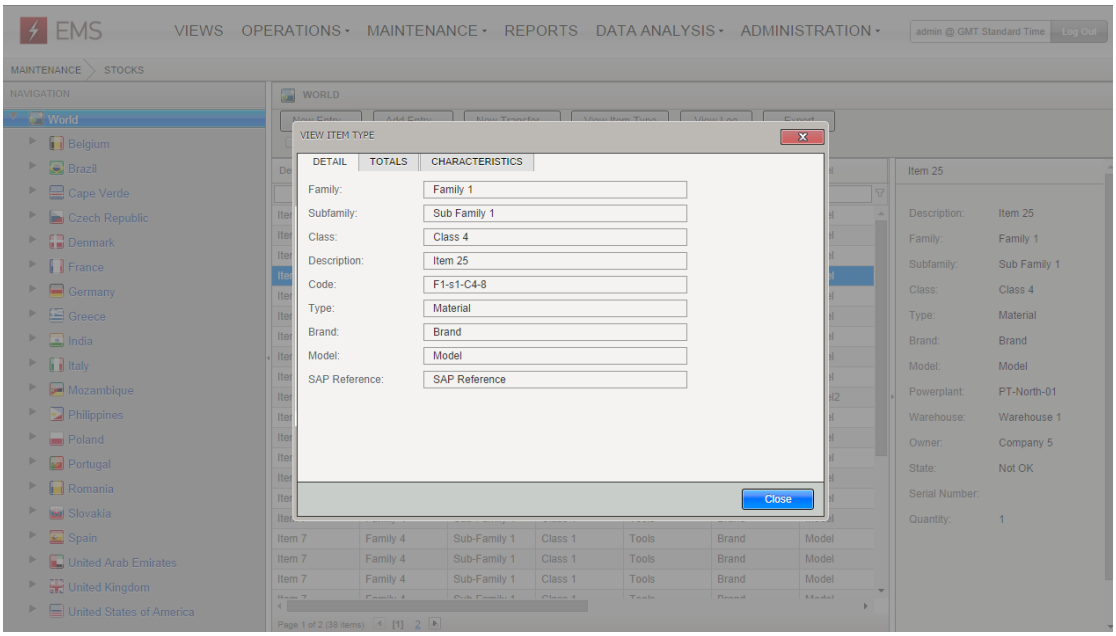


Figura 46 – popup view item type

- Popup view item type log (ilustrado na Figura 47): permite visualizar o registo de histórico de transferências de stocks de itens do tipo selecionado.

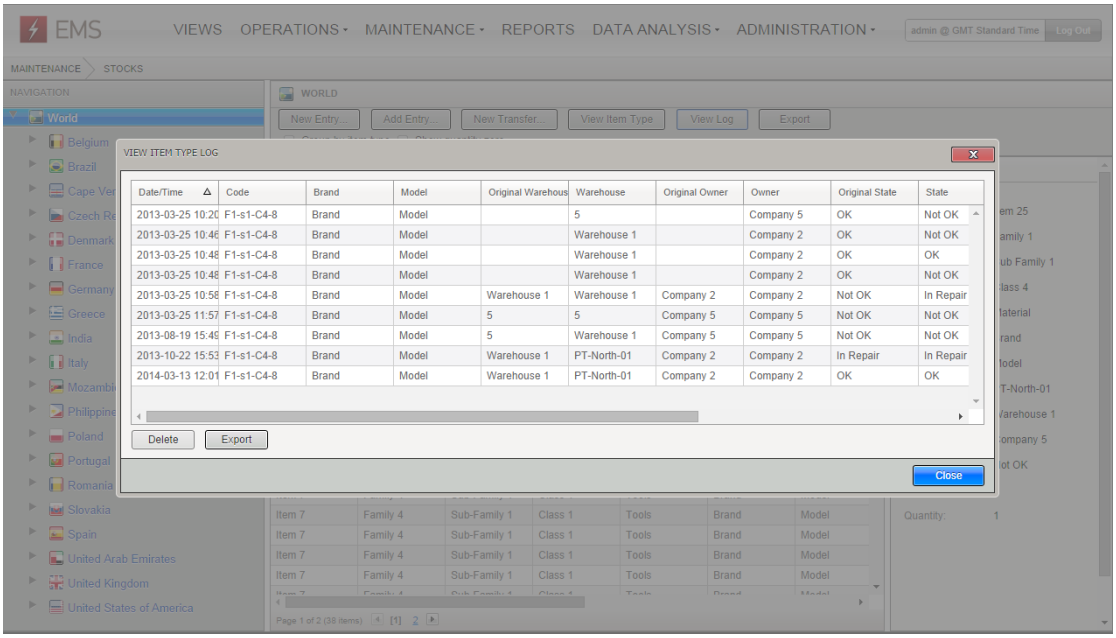


Figura 47 – popup view item type log

4.1.5.2 Descrição da análise de viabilidade da migração do módulo

Ao analisar o código relativo ao módulo selecionado, no sentido de proceder à análise de viabilidade da sua migração, rapidamente se constatou que existia um elemento que dificilmente tornaria essa migração viável para o âmbito deste trabalho.

Tipicamente a *framework* ASP.NET Web Forms disponibiliza um conjunto de controlos, em que cada um desses controlos é um objeto que pode ser personalizado em código, e que terá uma determinada correspondência em HTML. A título de exemplo, a Figura 48 ilustra a definição *markup* de um formulário, que dará origem ao HTML ilustrado na Figura 49.

```
<ol>
    <li>
        <asp:Label runat="server" AssociatedControlID="UserName">User name</asp:Label>
        <asp:TextBox runat="server" ID="UserName" />
        <asp:RequiredFieldValidator runat="server" ControlToValidate="UserName"
            CssClass="field-validation-error" ErrorMessage="The user name field is required." />
    </li>
    <li>
        <asp:Label runat="server" AssociatedControlID="Email">Email address</asp:Label>
        <asp:TextBox runat="server" ID="Email" />
        <asp:RequiredFieldValidator runat="server" ControlToValidate="Email"
            CssClass="field-validation-error" ErrorMessage="The email address field is required." />
    </li>
    <li>
        <asp:Label runat="server" AssociatedControlID="Password">Password</asp:Label>
        <asp:TextBox runat="server" ID="Password" TextMode="Password" />
        <asp:RequiredFieldValidator runat="server" ControlToValidate="Password"
            CssClass="field-validation-error" ErrorMessage="The password field is required." />
    </li>
    <li>
        <asp:Label runat="server" AssociatedControlID="ConfirmPassword">Confirm password</asp:Label>
        <asp:TextBox runat="server" ID="ConfirmPassword" TextMode="Password" />
        <asp:RequiredFieldValidator runat="server" ControlToValidate="ConfirmPassword"
             CssClass="field-validation-error" Display="Dynamic"
            ErrorMessage="The confirm password field is required." />
        <asp:CompareValidator runat="server" ControlToCompare="Password" ControlToValidate="ConfirmPassword"
             CssClass="field-validation-error" Display="Dynamic"
            ErrorMessage="The password and confirmation password do not match." />
    </li>
</ol>
```

Figura 48 – markup ASP.NET Web Forms

```
                    <ol>
                        <li>
                            <label for="MainContent_RegisterUser_CreateUserStepContainer_UserName">User name</label>
                            <input name="ctl00$MainContent$RegisterUser$CreateUserStepContainer$UserName" type="text"
id="MainContent_RegisterUser_CreateUserStepContainer_UserName" />
                            <span id="MainContent_RegisterUser_CreateUserStepContainer_ctl01" class="field-validation-error"
style="visibility:hidden;">The user name field is required.</span>
                        </li>
                        <li>
                            <label for="MainContent_RegisterUser_CreateUserStepContainer_Email">Email address</label>
                            <input name="ctl00$MainContent$RegisterUser$CreateUserStepContainer$Email" type="text"
id="MainContent_RegisterUser_CreateUserStepContainer_Email" />
                            <span id="MainContent_RegisterUser_CreateUserStepContainer_ctl03" class="field-validation-error"
style="visibility:hidden;">The email address field is required.</span>
                        </li>
                        <li>
                            <label for="MainContent_RegisterUser_CreateUserStepContainer_Password">Password</label>
                            <input name="ctl00$MainContent$RegisterUser$CreateUserStepContainer$Password" type="password"
id="MainContent_RegisterUser_CreateUserStepContainer_Password" />
                            <span id="MainContent_RegisterUser_CreateUserStepContainer_ctl05" class="field-validation-error"
style="visibility:hidden;">The password field is required.</span>
                        </li>
                        <li>
                            <label for="MainContent_RegisterUser_CreateUserStepContainer_ConfirmPassword">Confirm password</label>
                            <input name="ctl00$MainContent$RegisterUser$CreateUserStepContainer$ConfirmPassword" type="password"
id="MainContent_RegisterUser_CreateUserStepContainer_ConfirmPassword" />
                            <span id="MainContent_RegisterUser_CreateUserStepContainer_ctl07" class="field-validation-error"
style="display:none;">The confirm password field is required.</span>
                            <span id="MainContent_RegisterUser_CreateUserStepContainer_ctl08" class="field-validation-error"
style="display:none;">The password and confirmation password do not match.</span>
                        </li>
                    </ol>
                    <input type="submit" name="ctl00$MainContent$RegisterUser$CreateUserStepContainer$ctl09" value="Register"
onclick="javascript:WebForm_DoPostBackWithOptions(new
WebForm_PostBackOptions(&quot;ctl00$MainContent$RegisterUser$CreateUserStepContainer$ctl09&quot;, &quot;&quot;, true, &quot;&quot;,
&quot;&quot;, false, false))" />
                </fieldset>
```

Figura 49 – HTML gerado pelo markup ASP.NET Web Forms

No caso da *framework* ASP.NET MVC, nas vistas existem os HTML Helpers, que produzem o efeito dos controlos do ASP.NET Web Forms. Para cada chamada a um HTML Helper será criado o output de HTML correspondente. Por exemplo, a Figura 50 ilustra um bloco de uma vista MVC, que em execução dará origem ao HTML ilustrado na Figura 51.

```
<fieldset>
    <legend>Registration Form</legend>
    <ol>
        <li>
            <%: Html.LabelFor(m => m.UserName) %>
            <%: Html.TextBoxFor(m => m.UserName) %>
        </li>
        <li>
            <%: Html.LabelFor(m => m.Password) %>
            <%: Html.PasswordFor(m => m.Password) %>
        </li>
        <li>
            <%: Html.LabelFor(m => m.ConfirmPassword) %>
            <%: Html.PasswordFor(m => m.ConfirmPassword) %>
        </li>
    </ol>
    <input type="submit" value="Register" />
</fieldset>
```

Figura 50 – HTML Helpers em vista ASP.NET MVC

```
        <fieldset>
            <legend>Registration Form</legend>
            <ol>
                <li>
                    <label for="UserName">User name</label>
                    <input data-val="true" data-val-required="The User name field is required." id="UserName" name="UserName"
type="text" value="" />
                </li>
                <li>
                    <label for="Password">Password</label>
                    <input data-val="true" data-val-length="The Password must be at least 6 characters long." data-val-length-max="100"
data-val-length-min="6" data-val-required="The Password field is required." id="Password" name="Password" type="password" />
                </li>
                <li>
                    <label for="ConfirmPassword">Confirm password</label>
                    <input data-val="true" data-val-equalto="The password and confirmation password do not match." data-val-equalto-
other="*.Password" id="ConfirmPassword" name="ConfirmPassword" type="password" />
                </li>
            </ol>
            <input type="submit" value="Register" />
        </fieldset>
```

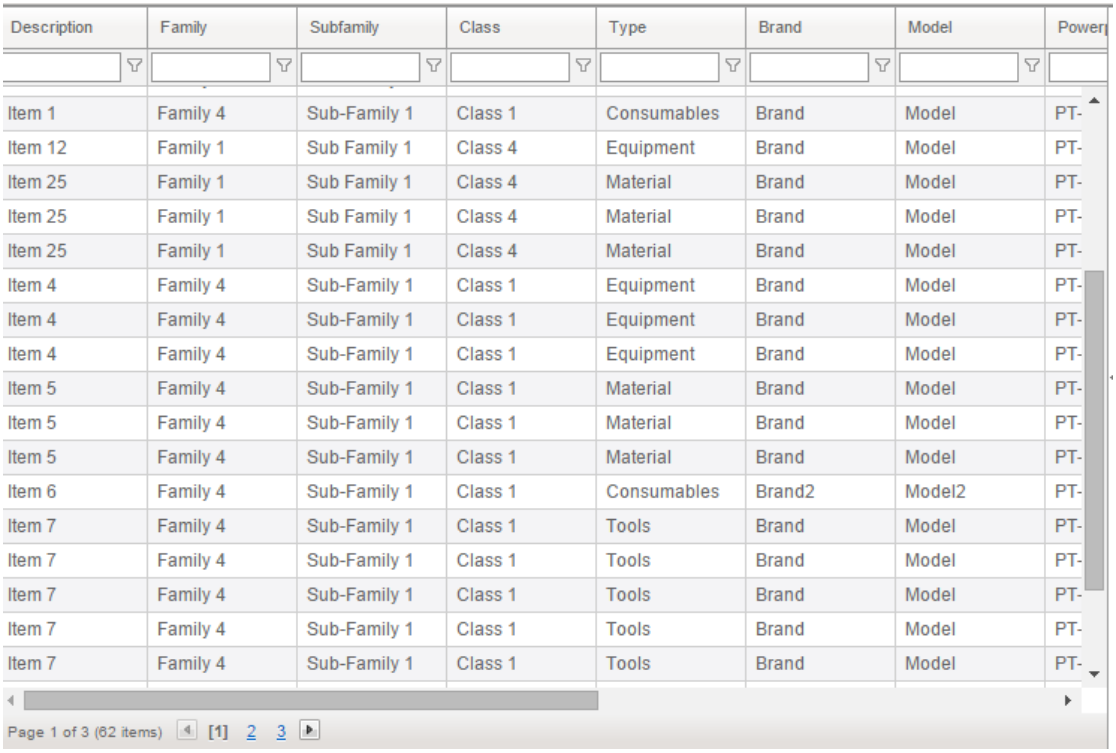
Figura 51 – HTML gerado pela vista ASP.NET MVC

Analisando os blocos de markup ASP.NET Web Forms e HTML correspondente, e vista ASP.NET MVC e HTML correspondente, rapidamente se constata que existe uma associação direta que pode ser feita entre os controlos do ASP.NET Web Forms, os HTML Helpers do ASP.NET MVC e o HTML correspondente. Por exemplo, o controlo Label é equivalente ao HTML Helper Html.LabelFor(), e ambos irão gerar um elemento do tipo label em HTML. Ou o caso do controlo TextBox que corresponde ao HTML Helper Html.TextBoxFor(), e ambos irão dar origem a um elemento HTML do tipo input.

No entanto, no caso do EMS são utilizados controlos diferentes dos controlos ASP.NET, nomeadamente a libraria de controlo DevExpress. Esta libraria possui um leque mais alargado de controlos que podem ser utilizados com a plataforma ASP.NET Web Forms, e que tipicamente são mais complexos, ao mesmo tempo que possibilitam funcionalidades mais ricas.

É aqui que se torna complicado comparar código do EMS com recurso a esta biblioteca de componentes, com código que eventualmente seria migrado, mas para o qual apenas teríamos à disposição os HTML Helpers por omissão do ASP.NET MVC. Existe funcionalidade que não conseguiríamos implementar apenas com recurso à plataforma e seus elementos por omissão.

Talvez o caso que mais salta à vista é o caso do controlo DevExpress GridView, responsável por implementar as vulgarmente conhecidas grelhas do site. Este controlo permite de forma completamente integrada e dependente da plataforma ASP.NET Web Forms implementar funcionalidades tão avançadas como filtragem, paginação de resultados ou edição *inline (*Figura 52*)*. Muitas vezes para ativar estas funcionalidades basta apenas definir uma propriedade do tipo booleano.



Description	Family	Subfamily	Class	Type	Brand	Model	Power
Item 1	Family 4	Sub-Family 1	Class 1	Consumables	Brand	Model	PT-
Item 12	Family 1	Sub Family 1	Class 4	Equipment	Brand	Model	PT-
Item 25	Family 1	Sub Family 1	Class 4	Material	Brand	Model	PT-
Item 25	Family 1	Sub Family 1	Class 4	Material	Brand	Model	PT-
Item 25	Family 1	Sub Family 1	Class 4	Material	Brand	Model	PT-
Item 4	Family 4	Sub-Family 1	Class 1	Equipment	Brand	Model	PT-
Item 4	Family 4	Sub-Family 1	Class 1	Equipment	Brand	Model	PT-
Item 4	Family 4	Sub-Family 1	Class 1	Equipment	Brand	Model	PT-
Item 5	Family 4	Sub-Family 1	Class 1	Material	Brand	Model	PT-
Item 5	Family 4	Sub-Family 1	Class 1	Material	Brand	Model	PT-
Item 5	Family 4	Sub-Family 1	Class 1	Material	Brand	Model	PT-
Item 6	Family 4	Sub-Family 1	Class 1	Consumables	Brand2	Model2	PT-
Item 7	Family 4	Sub-Family 1	Class 1	Tools	Brand	Model	PT-
Item 7	Family 4	Sub-Family 1	Class 1	Tools	Brand	Model	PT-
Item 7	Family 4	Sub-Family 1	Class 1	Tools	Brand	Model	PT-
Item 7	Family 4	Sub-Family 1	Class 1	Tools	Brand	Model	PT-
Item 7	Family 4	Sub-Family 1	Class 1	Tools	Brand	Model	PT-

Page 1 of 3 (62 items) [1] 2 3

Figura 52 – exemplo de grelha DevExpress

Não tendo ao meu dispor um componente equivalente em ASP.NET MVC, seria quase como reinventar a roda implementar este tipo de funcionalidade com os recursos disponíveis.

Mas mesmo analisando controlos mais simples, como por exemplo o vulgar botão, é possível verificar que existe um grande grau de complexidade no HTML gerado pelos controlos DevExpress, supostamente para implementarem funcionalidades mais avançadas. Vide por exemplo o HTML, acompanhado de código script, gerado automaticamente por um controlo tão simples como um botão, na Figura 53.

```
    <input class="crud-aspxbutton" id="newButton" value="New Entry..."
name="ctl00$ctl00$content$splitter$operationWorkArea$workAreaSplitter$toolboxPlaceholder$crudControl$newButton"
type="submit" style="font-family:Arial,sans-serif;font-size:12px;text-align:Center;vertical-align:Middle;" /><script
id="dxss_2080946189" type="text/javascript">
<!--

var dxo = new ASPxClientButton('newButton');
window['crudControl_newButton'] = dxo;
dxo.autoPostBack = true;
dxo.uniqueID =
'ctl00$ctl00$content$splitter$operationWorkArea$workAreaSplitter$toolboxPlaceholder$crudControl$newButton';
dxo.isNative = true;
dxo.Click.AddHandler(Stocks.OnNewEntry);
dxo.RegisterServerEventAssigned(['Click']);
dxo.InlineInitialize();

//-->
</script>
```

Figura 53 – HTML gerado por um controlo do tipo ASPXButton (DevExpress)

Mediante esta disparidade de comportamentos, qualquer comparativo seria de valor reduzido. Por exemplo, ao nível da performance seria difícil perceber se um eventual ganho em ASP.NET MVC se devesse à plataforma em sim, ou ao facto de deixarmos de usar componentes de terceiros que aparentemente são bastante pesados ao nível dos pedidos HTTP (e cujo eventual impacto na performance está fora do âmbito deste projeto).

A solução ideal seria obter uma biblioteca de componentes equivalentes para a plataforma ASP.NET MVC. E de facto ela existe, segundo o *site* do fabricante. No entanto exige licenciamento e um período de aprendizagem ainda grande, pelo que, e após avaliação, se descartou esta possibilidade de integração, pelo menos no âmbito deste trabalho.

4.2 Próximos passos

Neste trabalho foi feito um estudo e exercício de uma metodologia para evolução da arquitetura e plataforma de uma solução. No entanto, estudada uma forma de atingir esse objetivo, e após as ações executadas e documentadas neste relatório, interessa agora definir as ações necessárias a serem executadas no projeto do produto EMS, se pretenderem adotar a nova plataforma ASP.NET MVC.

4.2.1 Aquisição e integração da biblioteca de controlos DevExpress MVC

Como descrito na secção 4.1.5.2, foi detetada a necessidade de adquirir o licenciamento e integrar no projeto a biblioteca de controlos DevExpress MVC, por forma a garantir uma transição entre *frameworks* de suporte mais suave no projeto. Para cada controlo avançado que o EMS atualmente utilize desta biblioteca, como por exemplo as grelhas, existe o correspondente na biblioteca MVC (DevExpress Corporation, 2014).

Naturalmente que a equipa de projeto necessitará de um período de aprendizagem e adaptação aos novos componentes, mas este é um passo que me parece necessário se se decidir avançar para esta arquitetura.

Sugere-se que o trabalho a realizar na solução para integração desta biblioteca de controlos seja feita no *branch* criado para a realização deste projeto de análise, de forma a não interferir nas atividades recorrentes do projeto de produto (Figura 54).

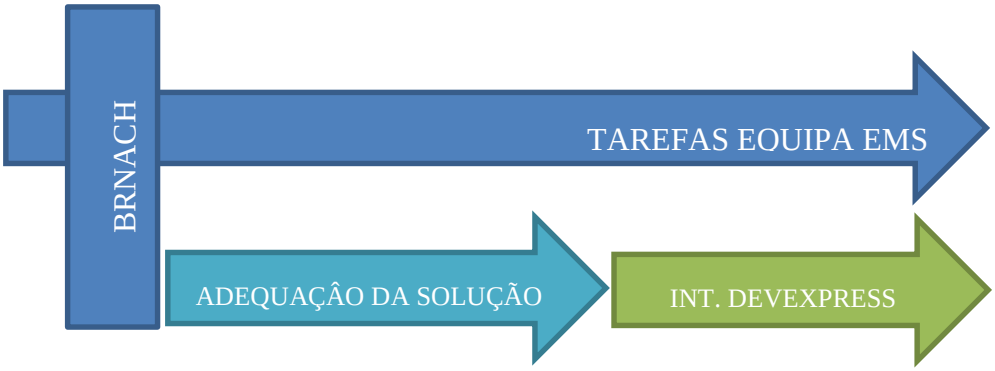


Figura 54 – tarefa de integração de biblioteca de controlos DevExpress

4.2.2 Criação de *master page* MVC

Executado este trabalho, a solução do EMS ficará dotada da capacidade de integrar de forma harmoniosa elementos das *frameworks* ASP.NET Web Forms e ASP.NET MVC. A partir dessa altura a equipa de desenvolvimento terá liberdade para implementar módulos futuros utilizando o padrão de arquitetura MVC, introduzido agora no projeto.

No entanto os módulos atuais do EMS partilham funcionalidade comum implementada numa ou várias *master pages*. É necessário implementar uma estrutura de *master pages* equivalente em MVC para poder implementar os novos módulos nesta plataforma.

Este trabalho poderá ser ainda demorado, mas apenas terá de ser executado uma vez, pois a ideia da *master page* é precisamente implementar de forma única todos os elementos repetidos em todas as páginas. Sugere-se que este trabalho seja executado num branch de desenvolvimento diferente daquele utilizado pela equipa de projeto, de forma a minimizar o impacto nas atividades recorrentes executadas pela equipa do produto neste período de transição (Figura 55).

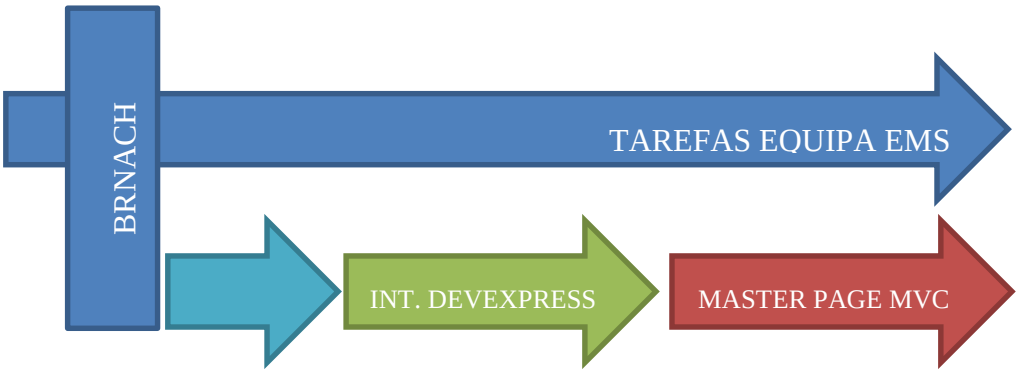


Figura 55 – tarefa de criação de master page MVC

4.2.3 *Merge* e início do desenvolvimento e manutenção em modo híbrido

Uma vez que foi criada uma ramificação no código (secção 4.1.1) sob o qual foram executadas todas as tarefas de adequação da solução descritas neste relatório, e onde ainda se sugere a realização dos pontos descritos em 4.2.1 e 4.2.2, será necessário em determinada altura convergir as ações efetuadas no código e configuração do IDE, para passar a serem acessíveis a toda a equipa de desenvolvimento os recursos da *framework* ASP.NET MVC. Deverá ser decidido qual o melhor momento para fazer essa integração, e nessa altura proceder à integração utilizando a técnica de *merge*.

Uma vez executada essa transição, passará a ser possível implementar os novos módulos utilizando a *framework* ASP.NET MVC, ao mesmo tempo que os módulos mais antigos podem ser mantidos sob a plataforma ASP.NET Web Forms (Figura 56).

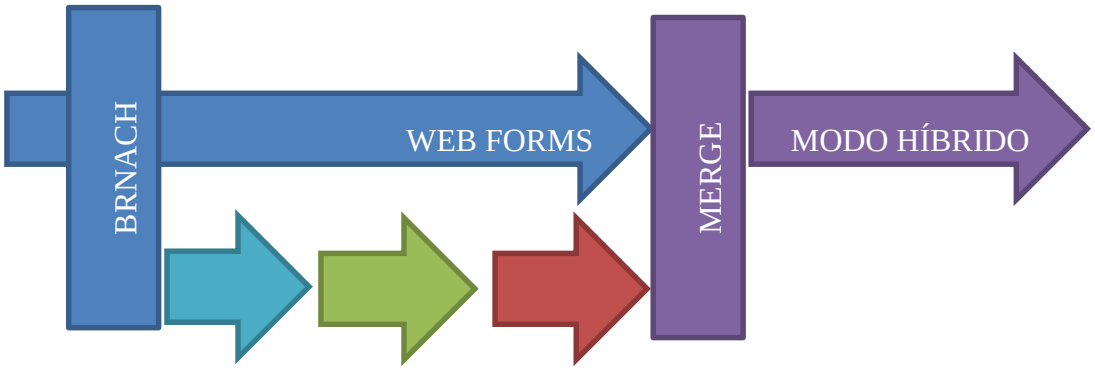


Figura 56 – tarefa de merge para integração de modo híbrido

4.2.4 Convergência para ASP.NET MVC

Nesta altura existirá uma duplicação de tecnologias e arquiteturas no projeto EMS, que dá à equipa de desenvolvimento do projeto a capacidade de utilizar uma *framework* bastante mais vocacionada para o desenvolvimento de produto, sem no entanto deixar de acarretar alguns potenciais problemas. O facto de existirem duas tecnologias pode-se tornar confuso, sobretudo para recursos menos experientes em ambas as tecnologias. O período de adaptação de novos recursos ao projeto deverá também aumentar (Stewa11, 2014).

Poderá ser interessante para o produto em questão tentar convergir todo o código do produto para a tecnologia mais recente, com o padrão de arquitetura MVC. Ao longo de um período de tempo médio ou longo existirão oportunidades para ponderar esta possibilidade. Por exemplo, se o número de bugs num módulo for em número suficientemente grande, porque não usar mais algum esforço e reimplementar na nova *framework*? Ou no caso de surgirem necessidades de alterações substanciais, porque não aproveitar a oportunidade para implementar de novo o módulo?

Esta convergência para apenas uma arquitetura deve ser ponderada, com o objetivo de, a médio ou longo prazo, eliminar definitivamente a arquitetura mais antiga do projeto, ASP.NET Web Forms e passar a ter um projeto 100% MVC (Figura 57). Obviamente que tal situação terá custos associados, mas poderão ser recuperados mais tarde.

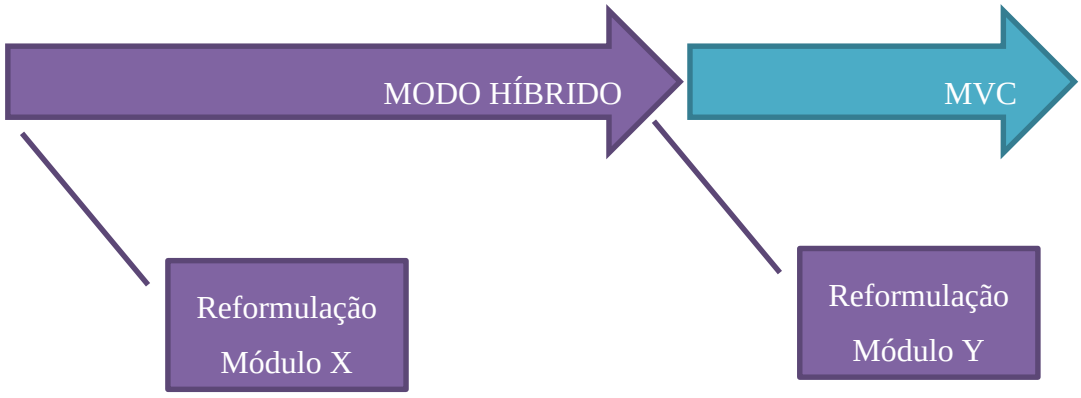


Figura 57 – convergência para arquitetura MVC

5 Análise de resultados

Uma vez descritos os fundamentos teóricos que serviram de suporte para este trabalho, a proposta de evolução tecnológica e a descrição do processo de adequação da solução, interessa agora discutir os resultados conseguidos. A interpretação dos resultados é uma visão claramente pessoal, mediante os objetivos propostos no início deste projeto e as experiências recolhidas durante a sua execução.

5.1 Adequação do trabalho à proposta inicial

Desde o momento da criação da proposta que deu origem a este trabalho até ao momento da sua entrega, ocorreram algumas mudanças que interessa referir, uma vez que tiveram impacto no resultado final.

A principal motivação para a realização deste trabalho foi sempre perceber até que ponto, para o produto de software em questão, se estaria a utilizar a arquitetura adequada. Desde o início da implementação do código que deu origem ao produto EMS, a arquitetura ASP.NET MVC ganhou força no universo do desenvolvimento em .NET, e apesar de ser opinião comum de que existem potencialmente várias vantagens em utilizar esta abordagem, nunca tinha sido feito um trabalho mais exaustivo de análise de adequabilidade para este produto em particular.

Durante o desenvolvimento deste trabalho percebi que, para além de comprovar que a arquitetura MVC traria vantagens ao produto e respetiva equipa de desenvolvimento, seria importante explorar a viabilidade de evolução da solução existente para esta arquitetura, no decorrer do ciclo de vida do projeto EMS. Sem esta ramificação no âmbito deste trabalho, este projeto poderia ser insuficiente para tomar uma decisão estratégica fundamentada sobre uma eventual transição de tecnologia. O estudo de um processo capaz de evoluir a tecnologia foi essencial para determinar o potencial impacto de uma evolução deste tipo, ao ser incluída numa equipa de produto com um *roadmap* definido, e tarefas de manutenção a serem executadas.

Foi portanto dado relevo a esta componente de processo, ainda mais potenciada pelo facto de, por limitações tecnológicas, não ter sido possível migrar na íntegra um módulo MVC existente. Mediante esta limitação, uma solução que permita que as duas

tecnologias coexistam torna-se mais importante, na medida em que no limite poderemos implementar novos módulos com recurso a esta nova tecnologia, coexistindo de forma transparente com os módulos existentes.

É portanto minha convicção de que esta pequena ramificação no âmbito deste projeto foi necessária, e acrescenta valor ao mesmo, face aos resultados obtidos.

5.2 Adaptação da solução à arquitetura MVC

Apesar de ser do interesse do projeto EMS conhecer as potenciais mais-valias desta arquitetura, é também importante apresentar o impacto que a estratégia de transição tecnológica poderá trazer, nomeadamente o esforço necessário e o ajuste que possa ser necessário nas tarefas de manutenção normais do projeto, assim como no *roadmap* de implementações futuras.

É do meu entendimento que qualquer solução que cause um grande desvio nas tarefas que estão a ser executadas diariamente no projeto dificilmente será implementada. Apesar de se conhecerem diversas oportunidades de melhoria na arquitetura da solução atual, trata-se sobretudo de um processo de otimização, uma vez que a solução atual cumpre os padrões de qualidade requeridos, e como tal esta transição deverá causar o menor impacto possível.

A migração integral dos módulos existentes para ASP.NET MVC é um processo bastante demorado, e como tal não encaixa no *roadmap* atual do produto. Não é portanto, a meu ver, uma solução válida para o projeto EMS, pelo menos no imediato.

A solução mais natural para encaixar neste cenário, e aquela que me parece mais viável face aos resultados apresentados, é a da evolução gradual do projeto para MVC ao longo de um período de tempo mais alargado. Em vez de se tentar migrar todo o código para a *framework* MVC, a ideia passa por tornar possível que os novos módulos e funcionalidades sejam implementados nessa tecnologia, ao mesmo tempo que os módulos existentes são mantidos na tecnologia atual.

A forma mais interessante de tornar as duas tecnologias compatíveis é, na minha opinião, fazê-las coexistir no mesmo IDE. Desta forma, para além do trabalho de manutenção dos módulos existentes e dos trabalhos de desenvolvimento de novos

módulos e funcionalidades ser mais facilmente executado, existe ainda o benefício de se poderem reutilizar componentes de código comuns. É facilmente percetível, comparando a arquitetura atual da solução, ilustrada na Figura 9, com a arquitetura proposta, ilustrada na Figura 26, que existem muitos componentes comuns, que podem e devem ser mantidos.

Nesse sentido, e apesar de não estar nos planos iniciais deste projeto, foi feita uma migração para Visual Studio 2013, uma vez que este IDE está claramente mais vocacionado para fazer coexistir as duas versões da *framework*. Ainda assim, para montagem da solução "híbrida" foi necessário alguma customização, de acordo com o documentado nas secções 4.1.2 e 4.1.3.

O resultado final foi a configuração de um ambiente de desenvolvimento capaz de dar suporte às funcionalidades desenvolvidas em Web Forms, ao mesmo tempo que existe liberdade para serem feitos novos desenvolvimentos em MVC. Este ambiente de desenvolvimento poderá facilmente ser integrado no ramo de desenvolvimento do projeto, e será a plataforma de suporte essencial a uma eventual evolução de tecnologias no EMS.

5.3 Desenvolvimento de novos módulos MVC

Uma vez montada a solução de desenvolvimento compatível com o novo padrão de desenho, é possível implementar novos módulos no projeto com recurso à *framework* ASP.NET MVC.

Neste trabalho apenas foi implementada uma pequena prova de que é possível implementar um módulo MVC na solução montada, uma vez que a completa implementação de um módulo nesta tecnologia não fazia parte do âmbito da análise. Ainda assim, a experiência adquirida durante a realização deste trabalho permite-me ter uma clara visão acerca do processo a seguir para a implementação de um novo módulo, integralmente escrito em MVC.

A grande maioria dos módulos atuais do EMS tiram partido de um conjunto de *master pages*, que implementam o *layout* base de todo o Portal Web. Cada elemento comum em vários módulos do portal, como por exemplo o menu de navegação ou o cabeçalho, são apenas implementados uma vez para todos os módulos, precisamente na *master*

page. O primeiro passo para a implementação do primeiro módulo é a implementação da *master page* MVC, utilizando o *view engine* ASP.NET, sob a qual irão ser exibidas as diversas vistas do módulo. Este passo deverá ser o mais intrusivo desta evolução no decorrer do ciclo de vida do produto, uma vez que a *master page* MVC implementada terá de implementar as funcionalidades já existentes na *master page* Web Forms. Terá ainda de se ter o cuidado de se manter o mesmo aspeto visual, algo que será facilitado pela partilha de CSS e recursos estáticas já existentes na solução do projeto. Como é óbvio, esta implementação apenas será necessária uma vez, e a partir dessa altura os módulos partilharão essa base comum.

A partir do momento em que temos uma *master page* MVC, o desenvolvimento dos módulos fica bastante simplificado, e não representará trabalho acrescido para além dos formulários do módulo propriamente dito. Uma vez optando pela implementação dos módulos em MVC, a equipa de projeto poderá tirar partido das vantagens da tecnologia descritas neste trabalho, como por exemplo uma mais clara divisão de código e a mais fácil integração de testes unitários.

5.4 Migração de componentes existentes

De forma a completar o processo de evolução do projeto EMS para uma arquitetura MVC, será necessário algures no tempo efetuar a migração dos módulos existentes. No entanto, a meu ver a migração dos módulos existentes não é essencial para o correto funcionamento do produto, podendo optar-se por nunca migrar totalmente para a nova arquitetura. As duas tecnologias poderão coexistir na mesma solução, inclusive partilhando alguns recursos comuns, como foi possível comprovar neste projeto. Há contudo sempre que ter em conta que enquanto o projeto tiver componentes nas duas arquiteturas, isso trará alguma complexidade acrescida. Por exemplo, um novo recurso do projeto terá uma curva de aprendizagem superior, uma vez que existem duas abordagens de implementação a serem estudadas.

Um ponto essencial a ter em conta se se optar por migrar os módulos existentes é encontrar uma biblioteca de componentes gráficos complexos para ASP.NET MVC. A migração deverá ser mais simples se se optar pelos componentes da mesma companhia, a DevExpress. Apesar da arquitetura dos componentes ser drasticamente diferente, as

capacidades e funcionalidades disponíveis e o estilo são claramente compatíveis, o que facilitará certamente a integração. Este é contudo um custo acrescido para o projeto, que deverá ser quantificado antes de tomar uma decisão a este nível.

O processo de migração dos módulos será contudo uma tarefa sempre demorada, e com custos para o projeto. No entanto, como o projeto suporta as duas tecnologias de forma integral, a eventual migração de um módulo poderá ser calendarizada de forma estratégica para o projeto. Uma forma de diminuir os custos associados à migração de um dos componentes existentes poderá ser, por exemplo, aproveitar oportunidades que surjam de alterações necessárias aos módulos existentes, ou a resolução de um grande volume de defeitos no mesmo módulo para migrar para a nova tecnologia.

6 Conclusões

Para realizar este trabalho foi feito um estudo das duas tecnologias de referência para a evolução tecnológica do EMS: a *framework* ASP.NET Web Forms, sob a qual se encontra implementada a versão atual do produto, e ASP.NET MVC, a plataforma para a qual se pretende estudar a evolução do produto no futuro.

Mediante o estudo das tecnologias envolvidas, com o conhecimento técnico adquirido sobre o produto EMS, foi possível identificar alguns pontos-chave em que uma transição de arquitetura poderá ser vantajosa. Existem, segundo a minha análise, melhorias a nível técnico com impacto direto no software, como por exemplo uma melhor performance, mas também claras vantagens em termos de arquitetura e modelo de desenvolvimento futuro do produto, como por exemplo a possibilidade de implementar testes unitários no código, ou a divisão mais clara das responsabilidades das classes, que poderá ser vantajosa, por exemplo, na manutenção do código e na distribuição de tarefas pelos elementos da equipa de desenvolvimento.

No sentido de adaptar o produto EMS ao ASP.NET MVC, foi desenhada uma nova arquitetura para o produto, de forma a integrar o máximo de elementos reutilizáveis possíveis da arquitetura já existente, uma vez que um dos meus objetivos foi sempre minimizar o *refactoring* necessário ao código para proceder a esta alteração.

Encontrada uma solução capaz de fazer o EMS funcionar sobre o padrão de desenho MVC, passou-se ao estudo de uma solução capaz de tornar esta integração no contexto real do projeto EMS, onde a migração integral do código existente está fora de questão, pelo menos no curto prazo, devido a restrições de tempo e recursos. A solução encontrada passou por encontrar uma solução "híbrida", capaz de conter as duas tecnologias, de forma a tornar possível uma evolução da arquitetura menos intrusiva no *roadmap* do produto EMS e nas tarefas da equipa de desenvolvimento.

Apesar da necessidade de se recorrer a alguma configuração de nível mais avançado, foi possível verificar que as últimas versões da ferramenta de desenvolvimento da Microsoft, o Visual Studio, possibilitam a coexistência em simultâneo das *frameworks* de desenvolvimento ASP.NET Web Forms e ASP.NET MVC. Esta abordagem possibilita que o projeto EMS, uma solução implementada em ASP.NET Web Forms já madura, possa passar a conter elementos de ASP.NET MVC, com o intuito de

proceder a uma migração de tecnologias progressiva, ao longo do tempo, de forma enquadrada com o *roadmap* do produto e as atividades diárias da equipa de desenvolvimento. Foi possível comprovar que a solução EMS passa a suportar a coexistência de componentes das duas plataformas, de forma transparente para o resultado final e experiência de utilização do produto.

Ainda assim, foram encontradas algumas limitações na solução atual que complicam um pouco a migração dos componentes ASP.NET Web Forms existentes para a arquitetura MVC. O principal elemento limitativo é o facto de ser usada uma biblioteca de controlos, da DevExpress, que apenas funcionam com a arquitetura Web Forms. Apesar de existir, do mesmo fabricante, uma biblioteca de controlos para ASP.NET MVC, existem necessidades de licenciamento associadas à utilização destes componentes, e teria ainda de considerar-se o tempo necessário para a curva de aprendizagem necessária. Este foi o principal motivo que limitou a realização da migração integral de um módulo, que por consequência impossibilitou uma comparação objetiva entre os resultados práticos da implementação dos módulos já existentes nesta nova arquitetura.

Foi possível, contudo, verificar durante a realização deste trabalho claras vantagens na utilização da nova arquitetura, com base na plataforma ASP.NET MVC. Melhor capacidade para realização de testes unitários e melhor divisão do código pelos elementos modelo, vista e controlador, que representam claramente melhorias na capacidade das equipas desenvolverem componentes e manterem o código desenvolvido.

Em resumo, pela realização deste trabalho fiquei com a opinião de é possível evoluir um projeto ASP.NET Web Forms para a vertente MVC da *framework* ASP.NET, e tirar daí vantagens competitivas. No entanto é necessário ponderar bem antes de avançar para essa transição. Existem custos associados a esta transição que apenas serão recuperados se se tirar verdadeiramente partido das vantagens desta tecnologia. Para tal é necessário por um lado verificar se existem os recursos necessários para proceder à transição, nomeadamente programadores com conhecimentos técnicos, tempo para proceder à adaptação da solução e migração de componentes e fundos para adquirir ferramentas ou bibliotecas de código. Existindo os recursos necessários, por outro lado interessa perceber se de facto se irá obter o retorno necessário. Para obter o retorno

necessário, deverá existir um *roadmap* suficientemente extenso, onde existam um número suficiente de alterações ou novos módulos e um calendário de manutenção suficientemente extenso para que as vantagens retiradas da nova arquitetura compensem o investimento inicial na transição.

Uma vez reunidas as condições descritas, sou da opinião que se deverá proceder a esta transição no projeto do produto EMS.

Bibliografia

Paz, José Rolando Guay (2013), Beginning ASP.NET MVC 4, Apress

Critical Software (2014), EMS, acedido pela última vez em Dezembro 2014 em: http://ems.criticalsoftware.com/

Critical Software (2014), Critical Software – dependable technologies for critical systems, acedido pela última vez em Dezembro 2014 em: http://www.criticalsoftware.com/

Vogel, Peter (2013), Migrating ASP.NET Web Forms to the MVC Pattern with the ASP.NET Web API, acedido pela última vez em Dezembro 2014 em: http://msdn.microsoft.com/en-us/magazine/jj991978.aspx

Helloco, Yann Le (2011), The Importance of Software Architecture, acedido pela última vez em Dezembro 2014 em: http://www.infovista.com/blog/index.php/2011/04/07/the-importance-of-software-architecture/

Yakima, Alex (2011), Design for Testability: A Vital Aspect of the System Architecture Role for SAFe, acedido pela última vez em Dezembro 2014 em: http://scaledagileframework.com/design-for-testability-a-vital-aspect-of-the-system-architect-role-in-safe/

Etheredge, Justin (2009), Building Testable ASP.NET MVC Applications, acedido pela última vez em Dezembro 2014 em: http://msdn.microsoft.com/en-us/magazine/dd942838.aspx

Reed, Dave (2006), TRULY Understanding ViewState, acedido pela última vez em Dezembro 2014 em: http://weblogs.asp.net/infinitiesloop/truly-understanding-viewstate

Strahl, Rick (2007), What's Ailing ASP.NET Web Forms, acedido pela última vez em Dezembro 2014 em: http://weblog.west-wind.com/posts/2007/Nov/27/Whats-Ailing-ASPNET-Web-Forms

DevExpress Corporation (2014), ASP.NET MVC, acedido pela última vez em Dezembro 2014 em: https://www.devexpress.com/Products/NET/Controls/ASP/MVC/

Galloway, Jon, Phil Haack, Brad Wilson, K. Scott Allen (2012), Professional ASP.NET MVC, John Wiley & Sons, Inc

Stewa11 (2014), Strategy considerations when migrating ASP.NET forms to MVC, acedido pela última vez em Dezembro 2014 em: http://www.codeproject.com/Articles/754763/Strategy-considerations-when-migrating-ASP-NET-for

Appel, Rachel (2013), Integrating ASP.NET Web Forms and ASP.NET MVC, acedido pela última vez em Dezembro 2014 em: http://rachelappel.com/integrating-aspnet-web-forms-and-aspnetmvc

Esposito, Dino (2014), Mixing Web Forms and ASP.NET MVC, acedido pela última vez em Dezembro 2014 em: https://www.simple-talk.com/dotnet/asp.net/mixing-web-forms-and-asp.net-mvc/

Anexos

Anexo A: documento de arquitetura do EMS

Apenas disponível em formato digital.