UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

DESENVOLVIMENTO DE APLICAÇÕES DE ESCALAS DE AVALIAÇÃO EM SAÚDE

Dissertação de Mestrado em Engenharia Biomédica

SÉRGIO FILIPE COUTINHO REBELO

Orientador: Professor João Agostinho Batista de Lacerda Pavão

Coorientador: Professor Luís José Calçada Torres Pereira

Vila Real, março de 2019

DESENVOLVIMENTO DE APLICAÇÕES DE ESCALAS DE AVALIAÇÃO EM SAÚDE

Dissertação de Mestrado em Engenharia Biomédica

Por

SÉRGIO FILIPE COUTINHO REBELO

Orientador: Professor João Agostinho Batista de Lacerda Pavão

Coorientador: Professor Luís José Calçada Torres Pereira

Dissertação submetida à

UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO

para obtenção do grau de

MESTRE

em Engenharia Biomédica, de acordo com o disposto no

DR – I série – A, Decreto-Lei nº 115/2013 de 7 agosto e no

Regulamento Geral dos Ciclos de Estudo Conducentes ao Grau de Mestre na UTAD

DR, 2.ª série – N.º 133 de 13 de julho de 2016

Dedico este trabalho aos meus pais e irmãos e a todos os meus amigos pelo apoio e motivação.

AGRADECIMENTOS

Ao longo da realização deste trabalho foram várias as pessoas que me dedicaram tempo, força e ensinamentos e que me ajudaram e permitiram chegar ao tão esperado produto final. De todas estas pessoas, quero destacar aquelas a quem devo um especial agradecimento.

Em primeiro lugar, quero agradecer toda a orientação, disponibilidade e dedicação do Professor João Agostinho Pavão. Agradeço profundamente todos os ensinamentos e valores que me transmitiu ao longo deste percurso. Agradeço também ao Professor Luís Torres Pereira por todo o seu apoio e orientação, que foram fulcrais para que conseguisse realizar com sucesso a presente dissertação.

Agradeço também ao Engenheiro Victor Costa do CHTMAD por todo o apoio nas questões da integração da aplicação no ambiente hospitalar. À Professora Rute Bastardo Pinto, pelo apoio nas questões da interface com o utilizador. E, finalmente, à SANOFI, pela ajuda e apoio em todos os aspetos clínicos envolvendo o Tromboembolismo Venoso.

Agradeço a todos os meus amigos pela paciência, carinho e motivação constante, e por todo o tempo que partilharam comigo durante esta fase, dando-me sempre a força necessária e nunca me deixando desanimar.

Por último, agradeço profundamente à minha família, em especial aos meus pais por me terem deixado vivenciar estes momentos e por me terem dado a possibilidade de crescer com eles. Obrigado pela educação e por todos os valores transmitidos. Um agradecimento especial também à minha irmã, que me acompanhou sempre desde o início. Por me ter ajudado e motivado sempre que necessário, pelo amor e por tudo o que me ensinou e, principalmente, por me ter ajudado a crescer a todos os níveis.

A todos, bem hajam!

RESUMO

As escalas são meios de determinação e estratificação de risco que permitem avaliar de forma rápida e eficiente o estado clínico dos utentes. A aplicação manual destas escalas, contudo, pode trazer consigo muitos problemas, pelo que a implementação eletrónica das mesmas em ambiente hospitalar, através de aplicações *web*, surge assim como uma melhor opção.

Um exemplo de uma patologia que requer um rápido diagnóstico e tratamento adequado é o Tromboembolismo Venoso (TEV), responsável por elevadas taxas de morbilidade e mortalidade nos hospitais, pelo que se torna crucial a estratificação do risco desta patologia, a qual pode ser realizada através do uso de escalas para avaliação e estratificação desse mesmo risco.

Neste trabalho, descreve-se o desenvolvimento de uma arquitetura de uma aplicação *web* para a implementação de escalas de risco. Esta arquitetura assenta num modelo cliente-servidor, contendo duas entidades distintas, um *Back-end* e um *Front-end*. Optou-se pelo uso do *Grails* como *Back-end* e do *Angular* como *Front-end*. Foram utilizadas nesta aplicação duas escalas de TEV, a escala de Caprini e a escala recomendada pela Associação Portuguesa de Cirurgia Ambulatória (APCA).

Palavras-chave: Escalas de avaliação de risco; Aplicação *web*; *Back*-end; *Front*-end; Tromboembolismo Venoso.

ABSTRACT

The scales are means of risk assessment that allow a quick and efficient evaluation of the clinical status of the patients. The manual application of these scales, however, can bring with it many problems, so that the electronic implementation of these scales in a hospital environment, through web applications, emerges as a better option.

An example of a pathology that requires a rapid diagnosis and adequate treatment is the Venous Thromboembolism (VTE), responsible for high rates of morbidity and mortality in hospitals, so that the risk stratification of this pathology is crucial, which can be performed through the use of scales for evaluation and stratification of this same risk.

In this work, we describe the development of an architecture of a web application for the implementation of risk scales. This architecture is based on a client-server model, containing two distinct entities, a Back-end and a Front-end. We have chosen to use Grails as Back-end and Angular as Front-end. In this application, two VTE scales were used, the Caprini scale and the scale recommended by the APCA (Associação Portuguesa de Cirurgia Ambulatória).

Keywords: Risk assessment scales; Web application; Back-end; Front-end; Venous thromboembolism.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

1. INTRODUÇÃO

1.1 Enquadramento geral

O registo eletrónico da informação clínica dos pacientes é atualmente possível graças às tecnologias existentes nos hospitais, no entanto este processo não é por si só suficiente para diagnosticar rápida e eficientemente uma situação de risco (Pavão, et al., 2017). Assim, é necessário recorrer a outro tipo de ferramentas de avaliação, como é o caso das escalas para determinação e estratificação do risco.

A aplicação manual das escalas de avaliação, no entanto, pode ser um trabalho demorado e exaustivo, sendo que será muito provável a ocorrência de erros, quer no preenchimento da escala, quer na realização dos cálculos finais, para além do facto da probabilidade de se perderem ser bastante elevada. Todas estas razões salientam a importância da implementação eletrónica destas mesmas escalas, processo este que irá facilitar muito o trabalho dos médicos e enfermeiros, tanto na aplicação da escala como no registo do seu resultado, diminuindo bastante a probabilidade de ocorrerem erros durante o seu preenchimento.

Assim, uma forma de contornar os problemas relacionados com aplicação manual de escalas é o desenvolvimento de aplicações *web* em ambiente hospitalar, campo este onde a Engenharia Biomédica assume um papel preponderante. Um engenheiro biomédico integra métodos e ferramentas das ciências, da tecnologia e da engenharia com objetivo de analisar e resolver problemas em medicina e na saúde em geral (Enderle & Bronzino, 2012). Deste modo, o engenheiro biomédico pode tornar-se fundamental no desenvolvimento e implementação de aplicações *web* nos hospitais, contribuindo assim para um melhor diagnóstico e avaliação de diferentes doenças.

Um conjunto de escalas cujo uso teria um grande impacto na saúde dos pacientes, está relacionado com o Tromboembolismo Venoso (TEV), patologia responsável por elevadas taxas de morbilidade e de mortalidade nos hospitais, sendo uma das principais causas de morte evitável em ambiente hospitalar (Cohen, et al., 2008; Alves, et al., 2015).

A implementação de escalas de TEV sob a forma eletrónica e integrada no processo de trabalho normal dos médicos e enfermeiros permite o rápido diagnóstico desta patologia, permitindo uma maior celeridade no seu tratamento e na determinação da tromboprofilaxia adequada.

1.2 Objetivos

A presente dissertação teve como objetivo principal o desenvolvimento de uma arquitetura de uma aplicação *web* para a implementação de escalas de risco em saúde.

Teve ainda como objetivo o estudo de um conjunto de tecnologias relacionadas com o desenvolvimento de aplicações *web* e respetiva integração noutros sistemas IT, como é o caso do Sistema Hospitalar. Em particular pretendia-se saber da possibilidade de desenvolver a aplicação numa modalidade *Full-stack* baseada em *JavaScript*.

Pretendia-se finalmente implementar um protótipo capaz de testar as componentes básicas da arquitetura, em especial as relacionadas com o sistema de informação a desenvolver.

1.3 Estrutura da dissertação

A presente dissertação está dividida em quatro capítulos principais, os quais são precedidos pelo capítulo da Introdução.

No âmbito da Introdução, faz-se um enquadramento geral do estudo sendo explicadas as razões da escolha da área temática, apresentando-se, de seguida, os objetivos do trabalho. O segundo capítulo reporta à importância da determinação do risco do Tromboembolismo Venoso e à necessidade do uso de escalas de avaliação, implementadas eletronicamente através de aplicações *web*, para a determinação desse mesmo risco. O terceiro capítulo apresenta e esclarece as tecnologias *web* necessárias para o desenvolvimento de aplicações *web*. No capítulo 4 é mostrada e detalhada a arquitetura e implementação da aplicação *web* para escalas de avaliação de risco. Por fim, no último capítulo, são apresentadas as conclusões retiradas do estudo e possíveis trabalhos futuros.

2. DETERMINAÇÃO DO NÍVEL DE RISCO DE TROMBOEMBOLISMO VENOSO

O registo de saúde eletrónico (EHR – *Electronic Health Record*) tem como objetivo reunir toda a informação essencial de cada cidadão para a melhoria da prestação de cuidados de saúde, permitindo o registo e partilha da informação clínica (SPMS, 2018). No entanto, apesar das tecnologias existentes nos hospitais permitirem o registo eletrónico da informação clínica dos pacientes, este processo não é só por si suficiente para diagnosticar de uma forma rápida e eficiente todas as situações de risco. Isto é, a informação clínica até pode estar presente, mas não é apresentada de uma forma clara e evidente de modo a que seja possível fazer a sua rápida interpretação e diagnóstico (Pavão, et al., 2017).

Deste modo, torna-se assim necessário recorrer a outro tipo de ferramentas de determinação de risco, ferramentas estas que podem ser baseadas na continuidade do histórico do paciente, ou numa só entrevista realizada perante uma determinada situação de saúde relacionada com o utente. O primeiro caso refere-se à implementação eletrónica de orientações clínicas e de sistemas de apoio à decisão clínica (CDS – *Clinical Decision Support*). Por outro lado, quando falamos em ferramentas baseadas no presente, referimo-nos às escalas para avaliação e estratificação de risco.

2.1 Sistemas de Apoio à Decisão Clínica

Os sistemas de apoio à decisão clínica são sistemas de tecnologia de informação na saúde com o objetivo de fornecer aos médicos e profissionais de saúde todo o suporte necessário para a tomada de decisões clínicas apropriadas. Contudo, este sistema apresenta ainda alguns problemas e desafios. Uma questão-chave deste processo é a velocidade e facilidade de acesso (Berner, 2009). Isto é, apesar de os clínicos reconhecerem a necessidade da informação, eles podem apenas decidir aceder à mesma se o puderem fazer de uma forma rápida e eficiente. Se, por outro lado, o acesso for difícil e demorado dificilmente os clínicos optarão por utilizar os sistemas CDS.

Outro grande problema dos sistemas CDS está relacionado com a quantidade e qualidade de informação recolhida, por forma a ser efetuado um conjunto de alertas úteis para os clínicos, mas que consiga diminuir ao máximo os alertas redundantes e inapropriados. Pois, um sistema que esteja constantemente a emitir alertas passará, muito provavelmente, a ser ignorado pelos profissionais de saúde (Pavão, et al., 2017). Assim, o aumento significativo

da informação recolhida nos EHR leva a que a eficácia e eficiência de um sistema CDS fique cada vez mais limitada (Yoon, et al., 2017).

Por outro lado, a sua implementação pode também ser um grande desafio, pois, a implementação da maioria dos sistemas CDS para as diferentes situações de risco hospitalar necessita de vários pontos de ligação ao EHR, havendo assim necessidade de que estes estejam preparados para os fornecer no formato e semântica corretos (Pavão, et al., 2017). No entanto, existem também alguns sistemas CDS que são independentes do EHR, o que pode constituir um grande desafio ao trabalhar com dois sistemas diferentes, uma vez que o facto do sistema CDS não estar integrado no EHR vai obrigar o médico a inserir a mesma informação duas vezes (no registo clínico e no CDS) (Berner, 2009). Outro desafio para a implementação dos sistemas CDS é que a cultura da medicina sempre destacou a autonomia individual do médico, o que leva a que as mudanças no sistema nem sempre sejam bem recebidas por parte dos mesmos (Berner, 2009). Assim, o facto de a implementação destes sistemas implicar um elevado número de alterações e adaptações nos métodos a que os clínicos estão normalmente habituados, pode na maior parte das vezes levar a uma não adesão por parte dos mesmos, por considerarem tratar-se de um sistema disruptivo das suas ações normais (Pavão, et al., 2017).

De notar ainda que este processo implica um aumento significativo de pessoal necessário, quer para a implementação, quer para a gestão e manutenção dos sistemas CDS. Por consequência, o aumento do número de funcionários leva a um aumento de custos (Berner, 2009; Berlin, et al., 2006).

Os sistemas CDS, quando bem desenvolvidos e implementados, possuem um grande potencial para melhorar a qualidade dos cuidados de saúde e até mesmo aumentar a eficiência e reduzir os custos. Contudo, para que os sistemas CDS sejam eficazes, os médicos devem estar motivados para o seu uso. Porém, todos os aspetos até agora referidos, como a velocidade e facilidade de acesso, a quantidade de informação e a autonomia na medicina, contribuem para a diminuição dessa motivação, o que leva a que os sistemas de CDS ainda estejam a lutar pela integração completa nos sistemas eletrónicos de saúde atuais (Yoon, et al., 2017).

2.2 Escalas de Determinação de Risco

Uma vez que os sistemas de CDS não estão completamente integrados nos sistemas eletrónicos, uma possível solução passará pelo recurso a escalas para avaliação e determinação de risco. As escalas de avaliação são instrumentos de ampla utilização clínica, muito usados na investigação e na prática de cuidados médicos, que facilitam a avaliação do estado clínico, permitindo observações muito mais sistematizadas e menos subjetivas. Estas são construídas e validadas estatisticamente de acordo com a população específica para que são alvo.

Existem várias escalas com aplicações em diferentes áreas da saúde, como por exemplo, na pediatria, para a avaliação do desenvolvimento e da saúde de crianças e jovens (Curado, 2016; Araújo, 2016; Bellman, et al., 2003). Na área da geriatria também são utilizadas bastantes escalas para a avaliação do estado de saúde dos idosos (Apóstolo, 2012), desde escalas de avaliação cognitiva, de avaliação do equilíbrio, de depressão e solidão, até escalas para avaliação do risco de desenvolvimento de úlceras de pressão (Hughes, et al., 1982; Silva, et al., 2007; Ferrari & Dalacorte, 2007; Ferreira, et al., 2007).

Adicionalmente, também existem escalas que avaliam propriedades específicas da saúde, como escalas que avaliam a dor (Grégoire & Finley, 2008; Stevens, et al., 1996), escalas que avaliam a depressão (Birleson, et al., 1987), escalas que avaliam a ansiedade (Spielberger, 1973), escalas que avaliam o stress (Cohen, et al., 1983), escalas que avaliam o risco de Diabetes (Silva, 2018), escalas que avaliam o risco de Tromboembolismo Venoso (Laryea & Champagne, 2013; Vaz, et al., 2012), escalas que avaliam o risco de lesões da pele em recém-nascidos e crianças (Huffines & Lodgson, 1997), escalas que avaliam as competências para alimentação oral (Thoyre, et al., 2007), entre muitas outras.

Embora as escalas de avaliação de risco possam ser consideradas de extrema importância para o diagnóstico e tratamento de várias doenças, estas apresentam, contudo, algumas limitações. A aplicação manual de uma escala adequada para um determinado paciente pode ser um trabalho exaustivo e demorado que vai obrigar o médico a consultar por inúmeras vezes o EHR, acabando todo este processo por ser desencorajador e consumir imenso tempo. Adicionalmente, o facto de serem feitas e preenchidas em papel traz com elas vários problemas como: maior probabilidade de se perderem, maior facilidade de ocorrência de erros, maior dificuldade na realização dos cálculos, maior dificuldade na consulta do histórico dos pacientes, para além do facto de estas não estarem integradas no fluxo de trabalho dos procedimentos médicos.

2.3 Implementação Eletrónica de Escalas

A tecnologia está cada vez mais desenvolvida no que ao setor da saúde diz respeito e, portanto, uma forma de combater os problemas e limitações das escalas de avaliação e de outras ferramentas utilizadas na saúde, é o desenvolvimento e implementação de aplicações *web* em ambiente hospitalar. As razões referidas na Secção 2.2, relacionadas com a aplicação manual de escalas de avaliação realçam a importância da implementação eletrónica de escalas para a avaliação e estratificação de risco. E é neste aspeto que a Engenharia, mais concretamente a Engenharia Biomédica, se torna fundamental. Tendo a tecnologia tido um impacto tão grande nos cuidados médicos, os profissionais de engenharia acabaram assim por se tornar intimamente envolvidos nos processos médicos. Como resultado deste impacto, a Engenharia Biomédica emergiu como um meio de integração para duas profissões dinâmicas, a medicina e a engenharia (Enderle & Bronzino, 2012).

A Engenharia Biomédica estabelece a interface entre a Engenharia e a Saúde, aplicando para isso os conhecimentos de Engenharia na análise e resolução das várias situações da medicina e da prática clínica. Isto é, combina conhecimentos de engenharia aplicados às necessidades médicas, a fim de disponibilizar novas técnicas e ferramentas (como biossensores, biomateriais, processamento de imagens e inteligência artificial) que os profissionais de saúde podem usar para pesquisa, diagnóstico e tratamento (Enderle & Bronzino, 2012). O engenheiro biomédico está envolvido em praticamente todos os aspetos do desenvolvimento de novas tecnologias médicas, com o objetivo de encontrar novas soluções para os problemas de saúde enfrentados pela nossa sociedade. Os engenheiros biomédicos podem assim fornecer as ferramentas e técnicas necessárias para tornar o sistema de saúde mais eficaz e eficiente, contribuindo para uma melhoria da qualidade de vida de todos nós (Bronzino, 2006).

Assim sendo, um aspeto onde o engenheiro biomédico pode intervir é na conceção de aplicações *web* e ferramentas de apoio à decisão implementadas e utilizadas nos hospitais, que têm como principal objetivo o diagnóstico, a avaliação e o tratamento das mais diversas doenças.

As escalas de avaliação podem ser implementadas e interpretadas eletronicamente, através de aplicações *web*, trazendo consigo muitas vantagens em relação às preenchidas manualmente. Estas vão facilitar o trabalho do clínico, quer na aplicação da escala, quer no registo do seu resultado. O seu acesso e preenchimento será muito mais facilitado, será também muito mais fácil e rápido consultar o histórico de cada paciente, os cálculos serão

feitos automaticamente e a possibilidade da ocorrência de erros será reduzida. Outra mais-valia que as aplicações *web* trazem consigo, é o facto de poderem ser implementadas e transportadas num telemóvel ou *tablet*, permitindo o seu fácil acesso e transporte. A implementação eletrónica de escalas assume-se assim como um processo não disruptivo que pode contribuir para uma boa integração no fluxo de trabalho de todos os profissionais de saúde (Pavão, et al., 2017).

No caso de uma instituição de saúde usar um número elevado de escalas, pode tornar-se mais simples de gerir a implementação de uma estrutura específica que acomode as respetivas aplicações e que permita o seu desenvolvimento e gestão, de um modo integrado, sob o ponto de vista de uma organização comum. O objetivo passa por ter uma infraestrutura integrada que permita, sempre que necessário, aceder e usufruir de aplicações de suporte a escalas.

2.4 Avaliação de Risco no TEV

2.4.1 Importância da Determinação do Risco de TEV

O Tromboembolismo Venoso (TEV) é uma patologia representada clinicamente pela Trombose Venosa Profunda (TVP) e pelo Tromboembolismo Pulmonar (TEP), que envolve praticamente todos os médicos, sendo transversal às diferentes especialidades médicas e cirúrgicas (Alves, et al., 2015). Surge como consequência de um processo inapropriado de coagulação do sangue, estando associado a complicações como síndrome pós-trombótico e hipertensão pulmonar crónica (Prandoni, et al., 1997), sendo responsável por elevadas taxas de morbilidade e de mortalidade nos hospitais (Beckman, et al., 2010; Cohen, et al., 2007; Cohen, et al., 2008), constituindo um importante problema de saúde. Posto isto, torna-se assim fundamental o seu rápido diagnóstico e tratamento adequado, quer imediato, quer a médio e longo prazo, com o objetivo de evitar possíveis complicações.

Um estudo realizado a nível mundial (estudo ENDORSE de Cohen, et al. (2008)), com o propósito de avaliar a prevalência do risco de TEV em doentes internados e determinar o número de doentes em risco que receberam uma profilaxia efetiva, revelou que uma grande proporção dos pacientes hospitalizados está em risco de TEV, mas existe uma baixa taxa de profilaxia apropriada. Estes dados vieram reforçar a necessidade do uso de estratégias hospitalares para avaliação do risco de TEV e a implementação de medidas que garantam que os pacientes em risco recebam uma profilaxia adequada. Os dados do estudo ENDORSE

relativamente a Portugal (França, et al., 2011) foram muito semelhantes aos globais, revelando as seguintes conclusões: mais de metade dos doentes internados está em risco de TEV e menos de dois terços recebem a profilaxia adequada. Neste contexto, os responsáveis alertaram para a exposição dos doentes a riscos desnecessários e para a necessidade de novas estratégias para a implementação de uma tromboprofilaxia venosa adequada nos hospitais portugueses. O mesmo relatório acrescenta ainda que, apesar de todas as recomendações e evidências em ambiente hospitalar, os médicos parecem continuar reticentes e tendem a subestimar o TEV como uma causa de mortalidade entre os pacientes, pelo que a tromboprofilaxia eficaz é frequentemente subutilizada. (Vaz, et al., 2012; Cohen, et al., 2008).

Assim, sendo o tromboembolismo venoso uma das causas de morte evitável mais comuns em pacientes hospitalizados (Alves, et al., 2015; Cohen, et al., 2008), torna-se crucial uma tromboprofilaxia adequada, com vista a uma melhoria efetiva do bem-estar dos doentes. Para tal, um fator importante para a prescrição da tromboprofilaxia é a estratificação do risco de tromboembolismo, a qual pode ser realizada através da utilização de escalas para a avaliação e estratificação desse mesmo risco. E, uma vez que é urgente a estratificação do risco de TEV, as escalas de avaliação de risco apresentam-se assim como uma boa opção (Pavão, et al., 2017).

2.4.2 Escalas de Avaliação de Risco de TEV

Para a avaliação do risco de Tromboembolismo Venoso, existem diferentes escalas que devem ser aplicadas consoante o estado clínico em que se encontra o paciente. Assim, a escala aplicada a um doente que acabou de sofrer uma cirurgia deve ser diferente, por exemplo, da escala utilizada para um doente oncológico, bem como a utilizada para uma grávida. De seguida, são apresentadas algumas das diferentes escalas utilizadas para determinar e avaliar o risco de TEV:

- Escala de Caprini (Caprini, 2005; Okuhara, et al., 2015), escala de Rogers (Laryea & Champagne, 2013) e escala elaborada pelo Capítulo de Cirurgia Vascular da Sociedade Portuguesa de Cirurgia (CCVSPC) (Vaz, et al., 2012), que avaliam o risco de TEV em pacientes cirúrgicos sujeitos a internamento.

- Escala da Associação Portuguesa de Cirurgia Ambulatória (APCA) (Pinho, et al., 2013), utilizada para os pacientes submetidos a cirurgia de ambulatório, isto é, pacientes que têm alta hospitalar no dia da intervenção cirúrgica ou no período máximo de 24 horas.

- Escala de Pádua (Barbar, et al., 2010) e Escala de Wells, ambas destinadas a pacientes clínicos hospitalizados. Esta última pode ser utilizada tanto para a determinação do risco de Trombose Venosa Profunda (TVP) (Wells, et al., 2003) como do risco de Tromboembolismo Pulmonar (TEP) (Wells, et al., 2001).

- Escala de Khorana (Verso, et al., 2012; Van Es, et al., 2017), desenvolvida para identificar pacientes oncológicos com alto risco de TEV, que podem ser elegíveis para tromboprofilaxia.

- Escalas de avaliação de risco de TEV durante a gravidez e puerpério (Nelson-Piercy, et al., 2015; Dargaud, et al., 2009), onde o risco da ocorrência de TEV é cerca de 2 a 4 vezes mais elevado do que em mulheres não grávidas com idade semelhante.

Para além da determinação do nível de risco de Tromboembolismo Venoso, as escalas também podem aconselhar a profilaxia mais adequada, mediante a estratificação do risco. Assim, as escalas de avaliação têm como objetivo obter dados que permitam a estratificação dos doentes relativamente ao risco de TEV e apurar as medidas de profilaxia. A profilaxia pode ser aplicada através de meios farmacológicos ou mecânicos, sendo que a profilaxia farmacológica oferece um maior grau de proteção. Por esta razão, a profilaxia através de meios mecânicos é geralmente aplicada juntamente com a profilaxia farmacológica, produzindo assim melhores resultados (Laryea & Champagne, 2013; Vaz, et al., 2012; Rose, 2018). Normalmente, a profilaxia farmacológica consiste na administração de heparina na dose e peso molecular correspondente ao grau de risco, podendo ser administrada por via subcutânea ou intravenosa. A profilaxia mecânica pode ser aplicada, por exemplo, através de: meias de compressão graduada (GCS), dispositivos de compressão pneumática intermitente (IPC), mecanismos de bombeamento venoso nos pés (VFP), entre outros (Kahn, et al., 2012).

No Centro Hospitalar de Trás-os-Montes e Alto Douro (CHTMAD), organização no contexto da qual se desenrolou o presente trabalho, são usadas, entre outras, as escalas de Caprini e APCA, tendo estas sido indicadas como referência, pelo facto de serem utilizadas neste hospital, em versão manual. Em relação à escala de Caprini, esta inclui um conjunto de fatores de risco que se encontram divididos em quatro grupos consoante o grau de risco, podendo representar cada indicador 1, 2, 3 ou 5 pontos. Quanto ao modelo APCA, apresenta os vários indicadores de risco distribuídos por três grandes grupos: fatores de risco “major”, risco cirúrgico e risco individual.

3. TECNOLOGIAS *WEB*

O uso de tecnologias *web* num ambiente com muitos utilizadores, como é o caso de um hospital, tem algumas vantagens significativas face a outras aplicações distribuídas (que não são *web*) e a aplicações do tipo *stand-alone*. Em relação a estas últimas, seria possível um funcionamento fechado, restrito a cada utilizador, mas tecnicamente torna-se inviável devido a problemas de manutenção (por exemplo, a atualização de versões, instalação individual, entre outros), mas também a problemas de integração com o sistema de informação subjacente à organização.

Por outro lado, o ambiente *web* em uso num hospital é devidamente controlado, pelo que oferece as condições de acesso e de segurança necessárias à sua utilização enquanto "middleware" entre um cliente e um servidor. Deste modo, usando uma arquitetura cliente-servidor estão facilitadas, em muito, quer as questões que se prendem com a manutenção quer os problemas de integração.

3.1 Aplicação *Web*

Uma aplicação *web* pode ser entendida como qualquer programa de computador "cliente-servidor" que utiliza navegadores e tecnologias *web* para executar tarefas através da Internet. Consiste num conjunto de sistemas informáticos projetados através da Internet que utiliza tecnologias *web* como HTML, CSS e *JavaScript*, podendo ser executado tanto a partir de um servidor HTTP, como localmente no próprio dispositivo do utilizador (Nations, 2018). É importante não confundir uma aplicação *web* com um *website*, uma vez que um *website* é um conjunto de páginas *web* com informações diversas que tem como objetivo informar ou exibir uma determinada informação, enquanto que, uma aplicação *web* funciona como uma espécie de sistema, em que é possível realizar muito mais ações do que num *website* normal.

Uma aplicação *web* funciona sob a arquitetura cliente-servidor, em que o processamento dos pedidos é realizado por dois lados: o servidor e o cliente. Este processo de comunicação é conhecido como solicitação e resposta, ou seja, o cliente gera uma solicitação para o servidor, que lhe responde, sendo o cliente representado pelo *browser* e o servidor representado pelo servidor *web* (Figura 1). O cliente solicita ao servidor um recurso, por exemplo, uma imagem ou uma página HTML, enquanto que o servidor *web* recebe essa mesma solicitação e devolve algo para o cliente (resposta) (FSW-CEULP, 2016; Palmeira, 2012).

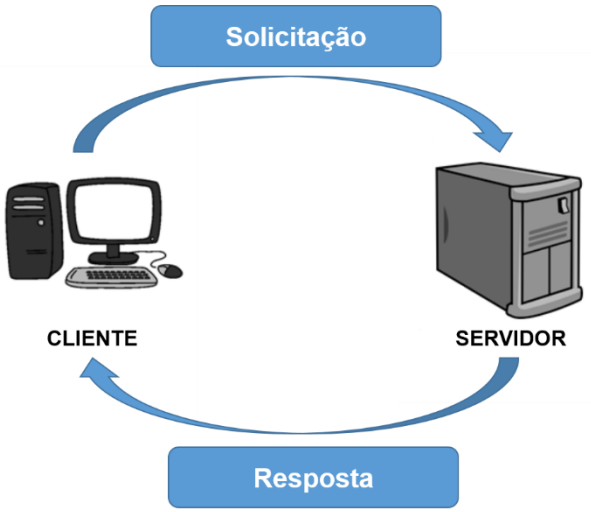

Figura 1. Sistema de comunicação das aplicações *web* conhecido como processo de "solicitação" e "resposta", baseado na arquitetura cliente-servidor.

A comunicação no ambiente *web* é feita através da Internet e, portanto, utiliza protocolos de comunicação de dados, sendo que o principal protocolo de comunicação na internet é o HTTP (*HyperText Transfer Protocol*), protocolo este que os clientes e servidores usam para se comunicar, sendo essa comunicação baseada em solicitações e respostas (FSW-CEULP, 2016). A solicitação HTTP possui outra solicitação conhecida como URL (*Uniform Resource Locator*), que é um recurso que se ativa quando o cliente tenta aceder aos métodos HTTP (Palmeira, 2012).

3.2 Modelo MVC

Muitos dos *frameworks web* utilizados seguem o padrão MVC (*Model-View-Controller*), que pode ser entendido como um padrão utilizado no desenvolvimento de *software* que separa a representação da informação (modelo) da interação do utilizador com o mesmo *software* (vista), ou seja, separa as representações internas de informação da forma como a informação é apresentada e aceite pelo utilizador (Reenskaug & Coplien, 2009). De uma forma resumida, o modelo é o componente central responsável pela leitura, escrita e validação dos dados; a vista consiste na exibição desses dados; e o controlador é responsável por receber todos os pedidos do utilizador, validando-os e convertendo-os em comandos para o modelo ou vista (Burbeck, 1992). O modo como estes três componentes se relacionam entre si está representado no esquema da Figura 2.

Este modelo é essencialmente usado para interfaces gráficas do utilizador (GUIs). Linguagens de programação como *Java*, C#, *Ruby*, *Python*, *Groovy* e PHP possuem *frameworks* MVC que são usados no desenvolvimento de aplicações *web*.

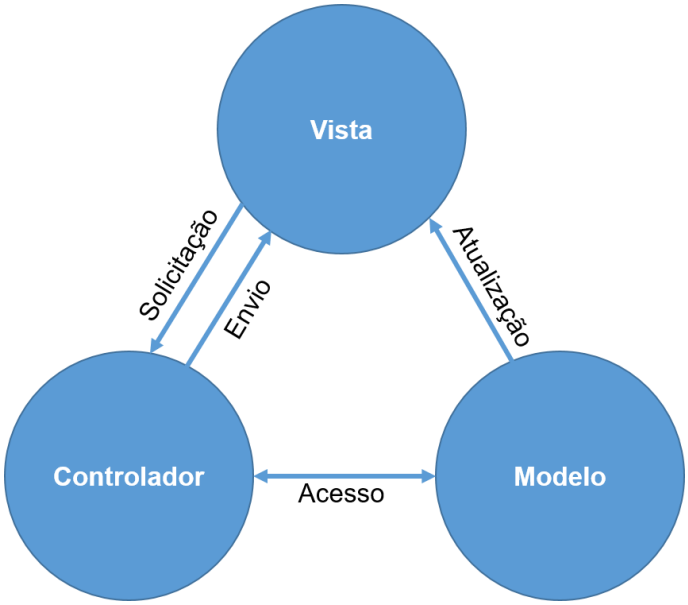

Figura 2. Arquitetura do padrão MVC.

3.3 *Frameworks Web*

Frameworks para aplicações *web* são *frameworks* de *software* que têm como principal função suportar o desenvolvimento de *websites*, aplicações *web* e serviços *web*, atenuando assim a sobrecarga associada às várias atividades de desenvolvimento *web*. Como já referido, a maior parte dos *frameworks web* são baseados no padrão MVC, uma vez que este promove a reutilização de código e permite a aplicação de várias interfaces.

Os *frameworks* são construídos para suportar o desenvolvimento de aplicações *web* baseadas numa única linguagem de programação, pelo que o número de *frameworks* atualmente utilizados na *web* é ainda considerável, sendo que estes podem ser de três tipos: *Front-end* ou "*user-side*", quando trabalham com a interface entre o sistema e o cliente; *Back-end* ou "*server-side*", quando se encarregam pelo funcionamento do *website*, gerindo a informação e processando os dados recebidos; ou então *Full-Stack*, quando englobam simultaneamente *Back-end* e *Front-end*.

Consideremos alguns exemplos de *frameworks* mais populares para o desenvolvimento *web* (VSCHART, 2018):

- *Frameworks* baseados em *Java*:

  - *Grails* ("*Groovy* on *Grails*") (Grails Framework, 2018), *framework open-source* utilizado para a construção de aplicações *web* através da linguagem de programação *Groovy*, que por sua vez é baseado na plataforma *Java*;

  - *Spring* (Spring Framework, 2018), *framework* de desenvolvimento *Java* que ajuda na criação de aplicações simples, rápidas e flexíveis baseadas em JVM (*Java Virtual Machine*);

  - *Play* (Play Framework, 2018), outro *framework* escrito em *Java* que segue a arquitetura MVC e que visa otimizar a produtividade do utilizador usando o paradigma "*Convention Over Configuration* (COC)".

- *Frameworks* baseados em *Node.js (JavaScript)*:

  - *Express* (Express Framework, 2018), *framework open-source* para *Node.js (JavaScript runtime)* (Node JS Framework, 2018) utilizado para projetos que usam *JavaScript* no *Back-end*;

  - *Meteor* (Meteor Framework, 2018), *framework Full-stack* suportado também em *Node.js* que utiliza *JavaScript* no *Back-end* e no *Front-end* e é usado para a criação de aplicações *web* em tempo real.

- *Frameworks* baseados em *Python*:

  - *Django* (Django Framework, 2018), *framework Full-stack* escrito em *Python* que segue uma arquitetura MTV (*Model-Template-Controller*) e que foi criado para desenvolver aplicações da forma mais rápida, versátil e eficaz possível para assim conseguir responder a prazos;

  - *Pyramid* (Pyramid Framework, 2018), *framework open-source* que também é baseado em *Python* e cujo seu principal objetivo é fazer o máximo possível com o mínimo de complexidade.

- *Frameworks* baseados em *Ruby*:

    - *Ruby on Rails* (Ruby on Rails Framework, 2018), *framework open-source* que utiliza a linguagem de programação *Ruby* e segue o padrão MVC, tendo sido criado para tornar a codificação de aplicações *web* mais simples e rápida.

- *Frameworks* baseados em PHP:

    - *Laravel* (Laravel Framework, 2018), *framework open-source* PHP que segue o padrão de arquitetura MVC e é destinado à criação de aplicações *web* de última geração;

    - *Yii* (Yii Framework, 2018), *Zend* (Zend Framework, 2018) e *Symfony* (Symfony Framework, 2018), são outros exemplos de *frameworks open-source* para PHP, usados para o desenvolvimento rápido de aplicações *web* em larga escala.

Na área da Saúde, uma das tecnologias mais usadas é o *Java*, pelo que iremos utilizar e descrever o *framework Grails* em maior detalhe.

O *Grails* veio reduzir a complexidade da construção de aplicações *web* na plataforma *Java*, baseando-se em tecnologias do ambiente *Java* já estabelecidas, como o *Spring* e o *Hibernate* (Grails Framework, 2018).

Na realidade, o *Grails* é suportado, na sua maior parte, em tecnologias já existentes, trazendo consigo um novo modo de trabalhar com estas. Assim, o *Grails* não é mais que um *framework Full-stack* baseado em MVC, utilizado para a construção de aplicações *web* e que se destina a ser um *framework web* de alta produtividade para a plataforma *Java*, seguindo para isso o paradigma de "*Convention Over Configuration* (COC)", ou seja, ao invés de utilizar uma série de arquivos XML, utiliza a COC para definir o papel das várias entidades de uma aplicação, diminuindo desta forma o número de decisões que o utilizador precisa de tomar, adotando como padrão algo que já é normalmente utilizado, uma "convecção" (Chen, 2006).

A linguagem de programação utilizada por este *framework* é o *Groovy*, linguagem dinâmica e produtiva cuja sintaxe é extremamente parecida com a do *Java*, onde a única diferença está nos recursos adicionais que esta linguagem oferece, resolvendo assim algumas inconveniências da linguagem *Java* de uma maneira bastante eficiente (Plohmann, 2012). Sendo baseado na plataforma *Java*, consequentemente, todo o código *Java* pré-existente pode ser usado de forma transparente inserido no código *Groovy*, uma vez que

ambas as linguagens compilam para *bytecodes* a serem interpretados por uma *Java Virtual Machine*.

3.4 Tecnologias *Web Front-end*

As tecnologias *web Front-end* especializam-se apenas pelas áreas que envolvem o utilizador, trabalhando apenas com a parte da aplicação que interage diretamente com o utilizador. Por outras palavras, caracterizam-se por "dar vida" à interface. Assim, o *Front-end* de um *website* é onde os utilizadores interagem com a página, é o que se vê quando se está a navegar na Internet, desde as cores aos menus.

Os desenvolvedores de *Front-end* são os responsáveis por toda a estrutura e arquitetura de interação do cliente com o *website*. Para tal, estes utilizadores necessitam de conhecer as três principais linguagens utilizadas em programação: HTML (linguagem de marcação), CSS (linguagem de estilo) e *JavaScript* (linguagem de programação). HTML (*Hyper Text Markup Language*) é a linguagem que vai exibir a informação e definir o conteúdo das páginas *web*, CSS (*Cascading Style Sheets*) é a linguagem que vai definir a apresentação dessa informação, incluindo o *design*, *layout* e variações de exibição, e, por fim, *JavaScript* é a linguagem que vai manipular e atribuir comportamentos a essa informação, sendo, portanto, responsável pela interatividade e comportamento das páginas *web* (W3schools.com, 2018).

Como exemplos das principais e mais usadas tecnologias *Front-end*, temos:

- *Angular* (ou *Angular2+*) (Angular Framework, 2018), plataforma *web* baseada em *TypeScript* e mantida pela Google, projetada para o desenvolvimento de aplicações *web* e aplicações móveis.

- *React* (ou *ReactJS*) (React Framework, 2018), uma biblioteca *JavaScript* criada e mantida pelo Facebook, sendo essencialmente usada para a criação de interfaces de utilizador para aplicações *web*.

- *Vue* (ou *Vue.js*) (Vue Framework, 2018), mais uma biblioteca *JavaScript* para a criação de interfaces com o utilizador. É um *framework open-source*, que, de certa forma, junta o melhor do *Angular* e do *React*, tendo sido a base do seu desenvolvimento os erros e sucessos destes dois.

Assim sendo, o facto de o *Angular* ser um *framework* da Google e uma das tecnologias *Front-end* mais utilizadas, leva a que seja esta a tecnologia *web* escolhida para desenvolver o presente estudo.

Nos últimos anos, o *Angular* tem crescido bastante como *framework* e como plataforma para o desenvolvimento de SPA (*Single Page Application*) e PWA (*Progressive Web App*) (Manjunath, 2018), de modo que, em 2016, foi lançada a tecnologia *Angular* (*Angular* 2+) (Angular Framework, 2018). Existia previamente uma outra versão do *Angular*, conhecida como *AngularJS* (*Angular* 1), a qual difere, em muito, das versões 2 e seguintes. Atualmente, o *Angular* já se encontra na versão 7.

Por sua vez, o *Angular* traz consigo uma longa lista de recursos que permitem construir desde aplicações *web* a aplicações móveis, usando sempre o padrão MVC. É caracterizado, essencialmente, pelo seu rápido ritmo de desenvolvimento e fácil integração de código, possuindo uma grande quantidade de recursos "*in the box*", o que leva a que não seja necessária uma preocupação com a instalação e atualização de muitos *plugins* (Angular Framework, 2018). A aplicação *Angular* tem uma arquitetura baseada em componentes que vão ser responsáveis por definir o conteúdo da página *web*, tendo cada componente, por sua vez, uma classe *JavaScript* e um *template* HTML/CSS associado.

Apesar de ser um *framework JavaScript*, o *Angular* recomenda o uso da linguagem *TypeScript* (TypeScript, 2018), desenvolvida pela Microsoft, e definida como um "superconjunto" estático de *JavaScript*, o que significa que, qualquer código válido de *JavaScript* também é um código válido de *TypeScript*. Na realidade, o *TypeScript* compila o código em *JavaScript*, uma vez que os navegadores não entendem a linguagem *TypeScript*, daí ser sempre necessário converter o código em código *JavaScript*. No entanto, esta tecnologia é capaz de oferecer alguns recursos adicionais que não estão presentes na versão atual de *JavaScript*, como por exemplo, programação orientada a objetos baseada em classes, interfaces, construtores e modificadores de acesso (público ou privado), atribuição opcional de tipos às variáveis, melhor verificação do código e maior deteção de erros comuns por parte dos IDEs (*Integrated Development Environments*) durante a escrita do código. (Manjunath, 2018).

3.5 *Web Services*

Uma solução bastante prática para solucionar a interoperabilidade entre sistemas e, consequentemente, garantir a sua comunicação, é o uso de *Web Services*, componentes

estes que permitem que aplicações possam enviar e receber dados através da Internet. Com esta nova tecnologia é, assim, possível que novas aplicações possam interagir com outras já existentes e que sistemas desenvolvidos em diferentes plataformas passem a ser compatíveis entre si. Isto porque, apesar de cada aplicação ter a sua própria linguagem, ela será traduzida para uma linguagem universal, um formato intermédio como o XML (*Extensible Markup Language*) ou o JSON (*JavaScript Object Notation*).

Deste modo, os *Web Services*, são utilizados para transferir dados através de protocolos de comunicação para diferentes plataformas, independentemente do tipo de plataforma e da linguagem de programação utilizada, permitindo desta forma a ligação entre diferentes aplicações que integram um determinado sistema. Assim, uma determinada aplicação pode invocar outra para efetuar determinadas tarefas, mesmo que as duas aplicações façam parte de sistemas diferentes e/ou estejam escritas em linguagens diferentes.

Assim sendo, a grande vantagem dos *Web Services* é que funcionam nos mais diversos sistemas operativos e diversos tipos de *hardware*. No entanto, a utilização de *Web Services* traz muitos mais benefícios, sendo os mais relevantes: a simplicidade na interoperabilidade de sistemas; a simplicidade na implementação (podem ser construídos em várias linguagens de programação); a maior segurança dos sistemas (não há um acesso direto à informação); a reutilização de código (o código é feito uma vez e pode ser utilizado posteriormente por diferentes serviços); e a redução de custos (tiram partido de protocolos e *frameworks web* já existentes, pelo que requerem pouco investimento) (Opensoft, 2017).

O funcionamento de um *Web Service* pode ser explicado da seguinte forma: uma determinada aplicação solicita uma das operações disponíveis no *Web Service* e, de seguida, este processa e envia os dados solicitados para a aplicação que, aquando da sua receção, interpreta-os e converte-os na sua própria linguagem (Figura 3).

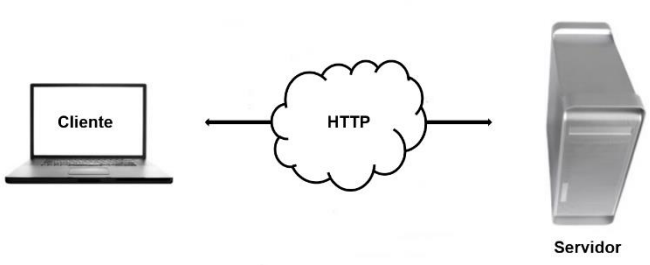

Figura 3. Modelo de funcionamento dos *Web Services*.

Os *Web Services* são identificados através de um URI (*Uniform Resource Identifier*) e a sua construção baseia-se, essencialmente, em duas tecnologias, XML e HTTP. Estes comunicam-se via protocolo HTTP ou HTTPS, sendo que enviam e recebem os dados no formato XML (Booth, et al., 2004; Erl, 2009). Ora, como mencionado anteriormente, para que linguagens diferentes possam comunicar entre elas é necessária uma linguagem intermédia que possa garantir a comunicação entre a linguagem do *Web Service* e do sistema que lhe faz o pedido. Para tal, existem protocolos de comunicação como é o caso do SOAP (*Simple Object Access Protocol*) e do REST (*Representational State Transfer*) (Booth, et al., 2004).

O protocolo SOAP baseia-se na linguagem XML para enviar mensagens e utiliza o protocolo HTTP para o transporte dos dados. Com isto, vai permitir que processos desenvolvidos em sistemas operativos diferentes possam comunicar entre si usando XML. Do mesmo modo como os protocolos da *web*, neste caso o HTTP, são executados em todos os sistemas operativos, o SOAP vai permitir que os clientes possam solicitar serviços da *web* e consigam receber as respostas independentemente da linguagem e das plataformas usadas (W3schools.com, 2018; Booth, et al., 2004).

O REST é um protocolo de comunicação mais recente que surgiu como alternativa aos *Web Services*, sendo baseado no protocolo HTTP e permitindo utilizar vários formatos de representação de dados, como o XML e o JSON, sendo este último o mais utilizado. A utilização do REST trouxe assim uma maior flexibilidade à comunicação entre sistemas, acompanhada da evolução dos padrões de arquitetura de *software* e da adoção de novos padrões (Erl, 2009; FSW-CEULP, 2016).

Com a utilização do REST, os verbos HTTP ganharam assim uma maior importância. Assim, quando o HTTP é usado, as operações disponíveis são GET, POST, PUT e DELETE. O GET é usado para solicitar dados, enquanto o POST é usado para enviar dados. Por sua vez, o PUT também é usado para enviar dados, sendo que a diferença entre estes dois últimos fica a cargo de cada API (*Application Programming Interface*). Por outro lado, o verbo DELETE é usado para excluir dados (FSW-CEULP, 2016).

Em geral, o SOAP é uma boa opção para instituições com ambientes rígidos e complexos (várias plataformas e sistemas). Nos restantes casos, a solução pode passar pelo uso do REST e JSON, uma vez que estas tecnologias são suportadas por praticamente todas as plataformas e linguagens existentes e apresentam uma solução final muito mais simples que o protocolo SOAP. Além disto, esta tecnologia exige menos experiência por parte do utilizador, ao contrário da outra tecnologia referida.

4. ARQUITETURA DE UMA APLICAÇÃO *WEB* PARA ESCALAS DE AVALIAÇÃO

4.1 Integração de uma Aplicação de Escalas num Ambiente Hospitalar

A implementação eletrónica de uma escala é, obviamente, uma aplicação. No entanto, que tipo de aplicação? Num ambiente hospitalar onde podem coexistir centenas de postos de trabalho faz sentido uma implementação distribuída, como é exemplo a arquitetura *web*. Esta arquitetura permite a distribuição de responsabilidades entre um cliente e um servidor, balanceando-as entre ambos os módulos. É importante diminuir o número de transações entre o cliente e o servidor por forma a ter um impacto o mais baixo possível no tráfego da rede, especialmente quando se trata de um número grande de aplicações a funcionar em simultâneo. Uma das formas de o conseguir é atribuir ao cliente mais responsabilidades podendo assim realizar mais ações de modo independente.

Por outro lado, à medida que o número de escalas a implementar aumente, encapsular cada uma delas numa aplicação pode não ser a melhor forma de gestão, uma vez que haverá um grande número de aplicações independentes a manter, quer na vertente do servidor, quer na vertente do cliente.

Assim, seria desejável diminuir o número de componentes a instalar e a manter. Uma das formas de o fazer será concentrar em um ou mais servidores toda a estrutura e escalas de valores, que depois podem ser acedidos por múltiplos clientes, os quais implementam as diferentes interfaces gráficas correspondentes às diferentes escalas. Será a decisão do gestor do Centro de Informática determinar o número de servidores e clientes por servidor em funcionamento. Esta arquitetura depende da definição de um sistema de informação suficientemente genérico que permita a implementação de diferentes escalas num mesmo servidor, debaixo de um só sistema de informação.

Outro requisito de integração está relacionado com a privacidade e confidencialidade dos dados. Pretende-se, portanto, que a troca de dados entre o *Front-end* e o *Back-end* seja realizada de uma forma segura.

Com o desenvolvimento acelerado que a *web* tem tido nos últimos anos torna-se simples usar um *Framework*, tal como o *Angular* ou o *React*, para desenvolver a parte do cliente, não só na sua componente gráfica como também na componente de interação com o

servidor e na realização de processamento autónomo. Uma aplicação deste tipo permite incorporar e gerir as regras de negócio, através do uso da linguagem *JavaScript*.

Como ponto de entrada na aplicação de escalas pretende-se um sistema não disruptivo da atividade clínica, sendo o SClinico (aplicação para EHR em uso na maior parte dos hospitais públicos e centros de cuidados primários) o que se afigura mais adequado para esta tarefa. O SClinico não é mais que um sistema de informação que vai permitir ir buscar informação sobre os pacientes e os médicos para usar na aplicação das escalas, servindo também como recipiente para escrever o resultado destas no respetivo ficheiro clínico do utente.

4.2 Metodologia e Proposta da Arquitetura

Começou-se por identificar e analisar as diferentes escalas de avaliação de risco utilizadas no Centro Hospitalar de Trás-os-Montes e Alto Douro (CHTMAD), a fim de selecionar as melhores escalas a serem utilizadas na presente aplicação. Por indicação do CHTMAD a escolha das escalas a envolver neste estudo recaiu sobre as escalas da TEV. Assim, foi adquirida e analisada toda a informação necessária acerca do Tromboembolismo Venoso, nomeadamente em que consiste esta patologia, a importância da prevenção do risco, as diferentes escalas de risco utilizadas e a tromboprofilaxia adequada. Posto isto foram selecionadas para implementação a Escala de Caprini e a escala da Associação Portuguesa de Cirurgia Ambulatória (APCA).

De seguida, após um estudo intensivo acerca das tecnologias, serviços e aplicações *web*, partiu-se para a definição da arquitetura de um sistema de escalas. O primeiro passo foi estabelecer um modelo de informação adequado às escalas a utilizar. Pretendia-se um modelo de informação que fosse abstrato, capaz de se constituir como uma estrutura genérica de informação para diferentes escalas a implementar futuramente.

O segundo passo foi averiguar qual o modelo de comunicação mais adequado, tendo como ponto de partida a condição de ter que ser um modelo cliente-servidor, tendo sido definida uma arquitetura com um *Back-end*, responsável por albergar toda a estrutura e conteúdo dos questionários e um *Front-end* para a gestão das regras dos questionários e a respetiva interface gráfica. Daqui resultou um diagrama de fluxo de informação entre os diferentes componentes da aplicação (Figura 4 e Secção 4.2.3).

Finalmente foi avaliado o modelo de comunicação externa, isto é, o modo de acolher na arquitetura pontos de ligação externos a fim de efetuar a integração no sistema hospitalar. Este modelo é apresentado na Figura 5 e descrito na Secção 4.2.4.

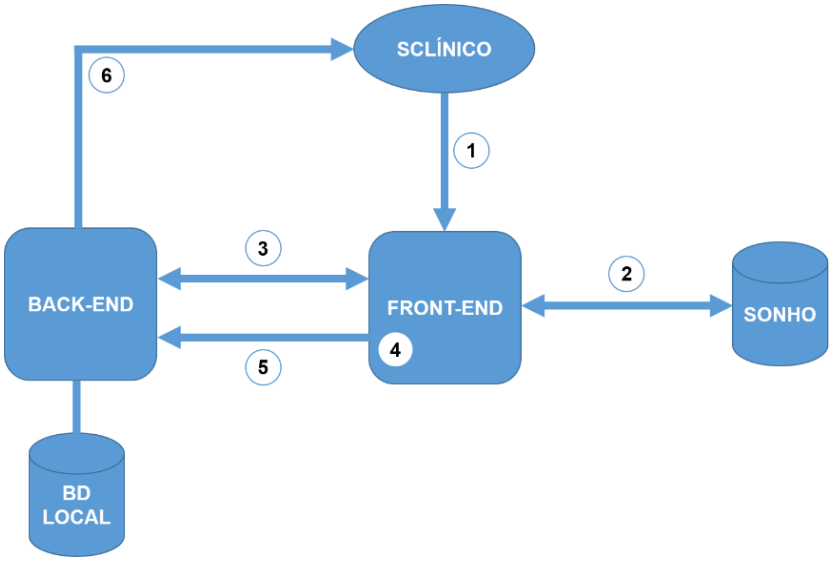

Figura 4. Diagrama de fluxo de informação entre os diferentes componentes envolvidos na aplicação.

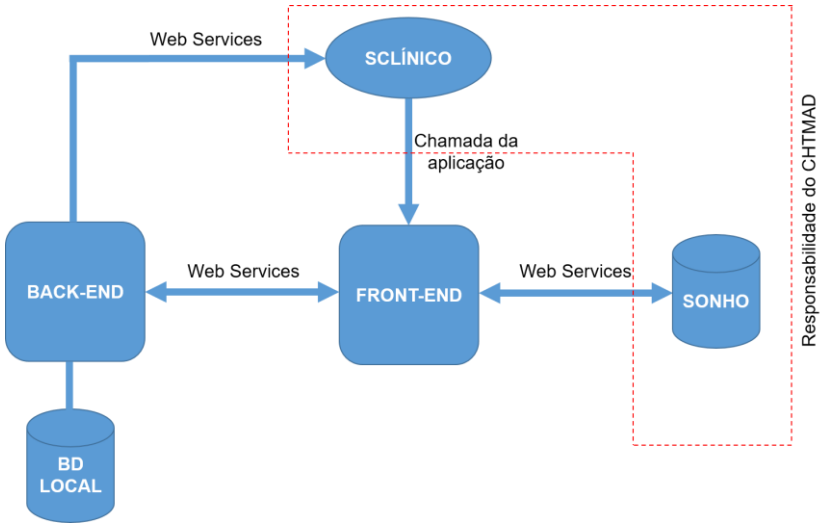

Figura 5. Modelo de integração da aplicação com o Sistema Hospitalar.

Para a implementação foi feito um levantamento de *frameworks web*, quer do lado do cliente quer do lado do servidor. Das muitas tecnologias hoje existentes, muitas delas *Full-stack*, optou-se pelo uso do *Grails* como *Back-end* e do *Angular* como *Front-end*, sendo a comunicação entre eles garantida por *Web Services*. O primeiro é baseado em tecnologia *Java*, correndo num servidor Tomcat, o que é importante para o ambiente hospitalar onde a

maior parte das aplicações corre com base nesta tecnologia. O segundo toma partido de um dos *frameworks* mais populares da *web* para a implementação de interfaces com o utilizador, com a assinatura da Google, baseado em *JavaScript*, o *Angular*.

4.2.1 Componentes do Sistema de *Back-end*

O sistema de *Back-end* é a entidade onde foi desenvolvida a estrutura das escalas de avaliação e definido todo o conteúdo pertencente a essas mesmas escalas. Assim sendo, foi criado um modelo de informação, genérico, adequado às escalas de avaliação, de forma a que esse mesmo modelo pudesse permitir a inclusão de vários questionários, e de várias escalas. Deste modo, adotou-se o modelo usado por Pavão, et al. (2017) onde foi adicionada mais uma classe às inicialmente definidas (Figura 6), passando assim a ser definidas quatro classes de domínio:

- *Question* – classe destinada às questões das escalas;
- *SubSection* – classe destinada ao agrupamento das questões em subsecções lógicas, isto é, de acordo com o conteúdo das diferentes perguntas;
- *Section* – classe destinada ao agrupamento das subsecções em secções lógicas do questionário;
- *Questionnaire* – classe destinada ao agrupamento das secções existentes.

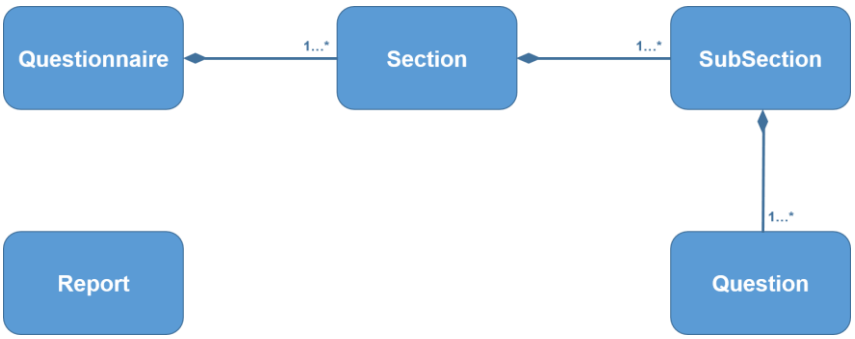

Figura 6. Modelo de informação genérico, adequado às escalas de avaliação.

A estas classes foi acrescentada outra, *Report*, que também se encontra na Figura 6 e que tem como função incluir as respostas assinaladas em termos de fatores de risco e sua respetiva profilaxia. De referir ainda que, em cada uma destas classes de domínio, podem ser definidos parâmetros, tais como título, descrição, valor e tipo. Tudo vai depender da informação que se necessitar para cada uma das classes de domínio, tornando desta forma a escala o mais completa e extensível possível.

Para além disto, neste sistema existe ainda um módulo para gerir uma base de dados de *backup* para os relatórios que contêm os resultados das escalas e registo de *logs*.

Finalmente, o sistema *Back-end* comporta também um módulo de envio dos relatórios para o SClínico.

4.2.2 Componentes do Sistema de *Front-end*

O *Front-end* é a entidade onde foram desenvolvidos os elementos de interação entre o sistema e o cliente mais adequados às diferentes escalas de avaliação. É a camada onde foram definidos todos os elementos gráficos responsáveis por "dar vida" à interface, como por exemplo, menus, botões, imagens, caixas de texto e cores.

Existe também um componente de regras onde, face às perguntas da escala que vão sendo assinaladas, efetua as contas do nível de risco e propõe a respetiva profilaxia adequada.

Este sistema contém ainda um módulo de impressão que permite ao médico, sempre que necessário, imprimir o resultado das escalas.

Por fim, outro componente importante que o sistema *Front-end* engloba é uma interface que é utilizada para a comunicação com o *Back-end*, e outra para a comunicação com o Sistema Integrado de Informação Hospitalar (SONHO), onde residem os dados dos pacientes.

4.2.3 Modelo de Comunicação entre os módulos envolvidos na Aplicação

Para o sistema estar integrado no dia a dia dos médicos e dos enfermeiros, o ponto de entrada na aplicação será o SClínico. Será a partir deste sistema que a aplicação de escalas será chamada (o médico clica num link integrado na interface do SClinico), enviando simultaneamente os dados sobre o médico e o número do processo do utente (ação 1 da Figura 4).

A partir do momento que a aplicação é chamada, através do número de processo, liga-se ao SONHO para obter os dados do paciente (ação 2 da Figura 4). Os sistemas *Back-end* e *Front-end* estão em comunicação, permitindo assim o carregamento do questionário específico da escala pretendida (ação 3 da Figura 4). De referir que estas duas últimas ações podem ser realizadas assincronamente.

No *Front-end*, é preenchida a escala e determinado o nível de risco e respetiva profilaxia (ação 4 da Figura 4). No final, os relatórios que contêm os resultados e logs são enviados para o *Back-end* ficando armazenados na base de dados de *Backup* (ação 5 da Figura 4).

Finalmente, o resultado da avaliação é enviado para o SClínico, de forma a ser integrado nos registos do paciente referentes àquele processo (ação 6 da Figura 4).

4.2.4 Modelo de Integração com o Sistema Hospitalar

Com o objetivo de efetuar a integração no sistema hospitalar, foi adotado o modelo presente na Figura 5, anteriormente apresentada.

Com base no presente modelo, a aplicação será inicialmente chamada a partir do SClínico. Aquando da sua chamada, o sistema *Front-end* liga-se ao SONHO através de *Web Services* (SOAP), passando ao mesmo tempo a comunicar com o sistema *Back-end*, sendo esta última ligação baseada em REST *Web Services*. De forma a garantir a comunicação entre a aplicação e a infraestrutura hospitalar é usado um componente *Web Services* baseado em SOAP, de modo a permitir o envio dos relatórios finais novamente para o SClínico.

Desta arquitetura, é importante referir que o desenvolvimento dos pontos de acesso da aplicação com o SClínico, assim como com o SONHO, é da responsabilidade do CHTMAD, ficando a cargo de quem desenvolve a aplicação das escalas o desenvolvimento dos respetivos clientes.

4.3 Implementação do Protótipo para Testes

Da arquitetura proposta apenas se implementou uma parte, uma vez que não houve tempo suficiente para a implementação completa. Por exemplo, os módulos de comunicação entre a aplicação e o sistema hospitalar implicam que, do lado do hospital, sejam fornecidos os respetivos pontos de acesso que, por razões de segurança, terão que ser testados

localmente, implicando reuniões e agendamentos de testes que não foi possível efetuar no tempo disponível. No entanto, a integração proposta por esta arquitetura já foi testada e validada num contexto muito próximo em Pavão, et al. (2017).

Deste modo, implementou-se o sistema de questionários, quer no *Back-end* quer no *Front-end*, para testar ambas as tecnologias envolvidas nos próprios componentes, mas também para testar as comunicações e a integração entre estes.

A implementação do protótipo foi feita usando o *framework Grails* (versão 3) como *Back-end* e a tecnologia *Angular* (versão 6) como *Front-end*. De forma a garantir a comunicação entre estas tecnologias, foi utilizado o protocolo REST *Web Services*, baseado em JSON. O uso de JSON tem vantagens sobre o XML nesta aplicação uma vez que se trata de uma aplicação *JavaScript*, do lado do *Front-end* e o *Grails* fornece de um modo simples respostas em JSON.

4.3.1 Implementação do Sistema de *Back-end*

Inicialmente, no *Grails*, foram definidas as classes de domínio segundo o modelo de informação presente na Figura 6. Em cada uma das classes foram definidas algumas variáveis, de acordo com a informação necessária para cada uma delas. Todas as variáveis utilizadas nas diferentes classes de domínio são exibidas na Figura 7. Simultaneamente, para que todas as classes estivessem associadas entre elas, de forma a que a estrutura do questionário permanecesse encadeada, foram utilizadas as relações *hasMany* (define a associação “um para muitos”) e *belongsTo* (define a relação “pertence a”). Adicionalmente, todas as classes de domínio foram definidas como um recurso REST, através da adição da transformação *Grails.rest.Resource*. Ao adicionar esta transformação e especificar um URI, as classes ficaram assim automaticamente disponíveis como um recurso REST nos formatos XML ou JSON (é possível definir estes formatos na configuração *Grails.mime.types* em *application.yml*). A mesma transformação registará automaticamente o mapeamento do RESTful URL necessário (Grails Framework, 2018).

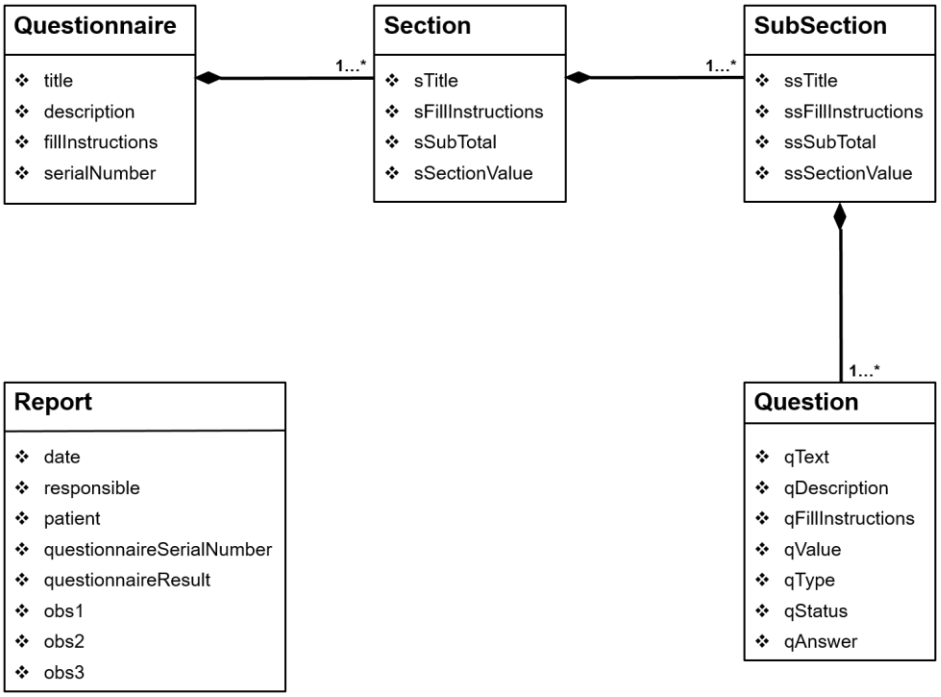

Figura 7. Modelo de informação do Sistema de *Back-end* com todas as variáveis definidas para cada uma das diferentes classes de domínio.

No que diz respeito aos controladores, estes foram criados contendo várias ações correspondentes aos verbos do HTTP usados pelos REST *Web Services*, que podem ser vistos, de seguida, na Tabela 1. Todos os controladores foram desenvolvidos em *Groovy*.

Tabela 1. Ações do *Grails* correspondentes aos métodos HTTP e respetivo URI, para o controlador *questions*.

Método HTTP	**URI**	**Ação do *Grails***
GET	/questions	index
GET	/questions/${id}	show
GET	/questions/${id}/edit	edit
POST	/questions	save
PUT	/questions/${id}	update
DELETE	/questions/${id}	delete

Em relação às vistas, estas foram desenvolvidas no formato JSON, uma vez que o perfil REST, por defeito, usa este formato de vistas para devolver respostas também elas em JSON. Estas desempenham um papel semelhante às GSP (*Groovy Server Pages*), mas são otimizadas para devolverem respostas JSON em vez de HTML. Para tal, foi necessário modificar o arquivo *build.gradle* para incluir o plug-in necessário para ativar as vistas JSON. Deste modo, foi então adicionado às dependências o comando "*compile org.Grails.plugins: views-json:1.2.7"*. Cada vista é alimentada por um controlador diferente de modo a permitir ao cliente solicitar informação separada das diferentes partes do questionário. Por exemplo, a ação *index()* no controlador *questions* devolve todas as questões existentes, sem o seu enquadramento devido no questionário. Para conseguir uma estrutura de questionário completa, procedeu-se à criação dos *templates*, de forma a encadear toda a estrutura do questionário no mesmo objeto JSON.

Para efeito de desenvolvimento não são usadas bases de dados físicas, o que permite uma maior rapidez nos testes. No entanto, e por este facto, sempre que há um arranque do servidor toda a informação contida na estrutura das classes de domínio é perdida. Uma forma de evitar este efeito é usar um ficheiro de Bootstrap (*Grails-app/conf/Bootstrap.Groovy*). Este ficheiro instancia todas as classes de domínio com a informação dos questionários reais e é sempre lido no arranque, preenchendo desta forma a base de dados "virtual" necessária aos testes. Aqui foram definidas todas as questões, subsecções, secções e questionários, sendo que as questões foram escolhidas e definidas de acordo com Caprini (2005) e Pinho, et al. (2013).

Durante a implementação foram configuradas algumas propriedades do *Grails* de forma a que o código pudesse funcionar corretamente e da forma pretendida, como por exemplo:

- Para que as classes criadas no *BootStrap.Groovy* fossem salvas da maneira mais preventiva e apropriada, ou seja, pudendo ser salvas mesmo quando a sua validação falhasse, foi necessário utilizar o comando "*save(failOnError:true)"*. Contudo, o *Grails* fornece esta opção de configuração, basta para isso acionar a propriedade "*Grails.gorm.failOnError:true"* em *Grails-app/conf/application.yml*.

- Ainda no arquivo *application.yml* é possível configurar o CORS (*Cross-Origin Resource Sharing*), através da adição da propriedade "*Grails.cors.enabled:true"*. O CORS é um mecanismo que usa cabeçalhos HTTP adicionais para permitir que um determinado utilizador tenha permissão para aceder ao servidor numa origem (domínio) diferente da do

website atual. Por motivos de segurança, os navegadores restringem solicitações HTTP de origens diferentes, ao passo que uma aplicação *web* só pode solicitar recursos HTTP do mesmo domínio da aplicação, a menos que sejam utilizados os cabeçalhos CORS (MDN Web Docs, 2018).

4.3.2 Implementação do Sistema de *Front-end*

A implementação do *Front-End* foi feita em *Angular*. Em primeiro lugar foram criados todos os componentes necessários, definidos na arquitetura, bem como as várias páginas *web* que fazem parte da aplicação, tendo sido introduzidos nestas os elementos essenciais para o seu funcionamento. Componentes como botões, cabeçalhos, menus e caixas de preenchimento vão fazer parte do conteúdo destas páginas *web*. A Figura 8 apresenta a visão geral das interfaces da aplicação TEV.

Figura 8. Página inicial da Aplicação TEV.

De salientar que todas as páginas e os seus componentes foram desenvolvidas em HTML e complementadas através das linguagens CSS e *TypeScript*, linguagens estas que vão definir a apresentação do conteúdo destas páginas e atribuir-lhe diferentes comportamentos, respetivamente. As CSS não foram codificadas diretamente, mas foram definidas recorrendo à tecnologia SASS (*Syntactically Awesome Style Sheets*), uma

linguagem de *script* que é interpretada ou compilada em CSS. As SASS permitem ter um maior controlo sobre a apresentação dos componentes tornando mais fácil de gerir as CSS para projetos maiores.

Assim, na página *web* inicial foram implementados, para além do cabeçalho e do componente dos dados dos pacientes, alguns botões, sendo que cada um destes redirecionava-nos para outras páginas diferentes, através da ação "*router*" do *Angular* em *src/app/app-routing-module.ts*. Nas páginas *web* seguintes foram introduzidos novos componentes, como o menu e o "accordion", elemento essencial que é responsável pela exibição ou ocultação das perguntas de cada questionário quando solicitado pelo utilizador. Todos os componentes utilizados ao longo da aplicação podem ser observados na Figura 9.

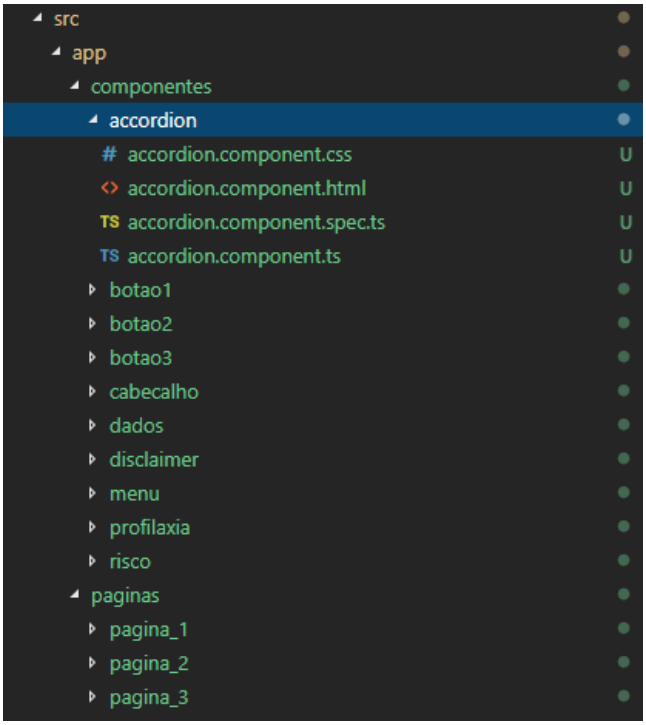

Figura 9. Componentes utilizados no desenvolvimento da aplicação na parte do *Angular*.

De todos os serviços *web* definidos na arquitetura apenas um era da nossa inteira responsabilidade. Foi, assim, criado um serviço de forma a ser possível carregar os questionários a partir do *Back-End Grails*. Para tal, recorreu-se ao método GET, que é a operação do REST responsável pela solicitação dos dados. O código utilizado neste *Web Service* do lado do *Angular* pode ser consultado na Figura 10.

```
@Injectable()
export class QuestServerService {
  protocol: string = 'http';
  hostIp: string='localhost';
  port: string = '8080';

  baseString: string = this.protocol+'://' + this.hostIp + ':' + this.port + '/';



  constructor(private http: Http) {}

  getQuestionnairesGrails(questionnaireName: string, questionnaireNumber: number) {

    let s = this.baseString + questionnaireName + '/' + questionnaireNumber;
    console.log(s);

    return this.http.get(s)
    .toPromise()
    .then(res => res.json())
    .catch(this.handleError)
  }

  private handleError(error:any): Promise<any> {
    return Promise.reject(error.message || error);
  }

}
```

Figura 10. Código do *Web Service* desenvolvido do lado do *Front-end* (*Angular*).

De salientar também que foram usadas funções assíncronas, permitindo escrever o código sem bloquear o segmento principal, ou seja, o código pode ir sendo executado sem ter necessariamente de esperar que outros processos terminem. Por sua vez, quando uma função assíncrona é chamada, ela retorna sempre uma promessa, esta que pode ser atendida ou rejeitada. Isto é, uma promessa pode ser entendida como um objeto que representa o resultado de uma operação assíncrona, podendo este resultado ser bem-sucedido, originando a chamada do método "*then*", ou mal-sucedido, provocando a chamada do método "*catch*".

5. CONCLUSÕES E TRABALHO FUTURO

Esta dissertação apresenta uma aplicação TEV constituída por duas escalas para a avaliação e estratificação do risco de Tromboembolismo Venoso, a escala de Caprini e a escala da APCA. A arquitetura da aplicação foi desenvolvida com o objetivo de disponibilizar aos clínicos, de uma forma o mais possível integrada no seu *workflow*, todos os métodos e ferramentas necessárias para a avaliação e determinação do risco de TEV e prescrição da tromboprofilaxia adequada aos pacientes que sofrem desta patologia.

Como primeiro objetivo deste trabalho estava definido o estudo de um conjunto de *frameworks web*, *JavaScript*, capazes de implementar estruturadamente interfaces com o utilizador. Para cumprir este objetivo procedeu-se ao levantamento das diferentes *frameworks* existentes para este fim, com especial relevo para o *Angular*, por indicação da equipa de orientação.

Embora todo o desenvolvimento da aplicação pudesse ser feito em *JavaScript*, recorrendo ao *Node.js* (implementação *Full-stack*), este *run-time* ainda não oferece condições de confiança suficientes para uso num ambiente mais conservador e intensivo como no caso de um hospital. Assim, desenvolveu-se a aplicação em duas entidades: um *Back-end* desenvolvido para o ambiente *Java*, utilizando o servidor Tomcat e uma parte de *Front-end* suportada em *Angular*.

A arquitetura proposta não pode ser testada na sua plenitude devido sobretudo à morosidade do agendamento de um teste de integração no ambiente real do hospital. O tempo investido no estudo das tecnologias e no *setup* técnico dos componentes contribuiu também para uma limitação do tempo disponível para a implementação completa da arquitetura.

Outro aspeto, que não dependia de nós, era o desenvolvimento dos *end-points* de ligação ao SONHO e ao SCLÍNICO. No entanto, baseando-nos nos testes efetuados por Pavão, et al. (2017) consideramos que se trata apenas de uma questão elementar de tempo e não uma questão técnica de fundo.

Assim foram implementadas as funcionalidades nucleares do sistema de *Back-end* e de *Front-end*, bem como a comunicação entre eles. O investimento central foi feito nesta implementação, em detrimento de outras partes, à partida mais simples, relacionadas com a interface com o utilizador. Os questionários foram corretamente implementados no *Back-end*. Foi ainda implementado, neste sistema, o *Web Service* que vai atender os pedidos de

questionários realizados pelo *Front-end*. Esta comunicação foi efetuada com sucesso, sendo as solicitações feitas ao servidor atendidas e corretamente recebidas e interpretadas por parte do cliente.

Apesar de menos trabalhados os aspetos da interface gráfica com o utilizador, toda a tecnologia envolvida nesta interface foi colocada em posição e testado o seu funcionamento, com sucesso, através de interfaces de *mockup*.

Para trabalho futuro, sugere-se um maior investimento nos aspetos gráficos de interatividade com o utilizador, assim como o desenvolvimento e testes de integração em ambiente real, no hospital, envolvendo os pontos de acesso com o SONHO e o SCLINICO. Por falta de tempo, o requisito da segurança não pode ser tratado, pelo que se sugere a execução desta tarefa no futuro.

REFERÊNCIAS BIBLIOGRÁFICAS

Alves, C., Almeida, C. & Balhau, A., 2015. *"Capítulo de Cirurgia Vascular" in Tromboembolismo Venoso Diagnóstico e Tratamento.* Sociedade Portuguesa de Cirurgia.

Angular Framework. *One framework. Mobile & desktop.* [Online] Available at: https://*Angular*.io/ [Acesso em fevereiro 2018].

Apóstolo, J. L. A., 2012. *Instrumentos para Avaliação em Geriatria (Geriatric Instruments).* Coimbra: Escola Superior de Enfermagem de Coimbra.

Araújo, S. I. M., 2016. *A avaliação do desenvolvimento infantil: relevância na intervenção do enfermeiro especialista em saúde infantil e pediatria.* Lisboa: Escola Superior de Enfermagem de Lisboa. .

Barbar, S. et al., 2010. *A risk assessment model for the identification of hospitalized medical patients at risk for venous thromboembolism: the Padua Prediction Score.* Journal of Thrombosis and Haemostasis, 8(11), 2450–2457.

Beckman, M. G., Hooper, W. C., Critchley, S. E. & Ortel, T. L., 2010. *Venous Thromboembolism.* American Journal of Preventive Medicine, 38(4), S495–S501..

Bellman, M., Lingam, S. & Aukett, A., 2003. *SGS II Escala de Avaliação das Competências no Desenvolvimento Infantil dos 0 aos 5 anos.* Lisboa: CEGOC-TEA.

Berlin, A., Sorani, M. & Sim, I., 2006. *A taxonomic description of computer-based clinical decision support systems.* Journal of Biomedical Informatics, 39(6), 656-667.

Berner, E. S., 2009. *Clinical decision support systems: state of the art.* AHRQ Publ..

Birleson, P., Hudson, I., Grey-Buchanan, D. & Wolff, S., 1987. *Clinical Evaluation of a SelfRating Scale for Depressive Disorder in Childhood (Depression Self-Rating Scale).* Journal of Child Psychology and Psychiatry, 28, 43-60.

Booth, D. & al, e., 2004. *Web Services Architecture.* World Wide Web Consortium. cap.3.

Booth, D. et al., 2004. *Web Services Architecture.* World Wide Web Consortium.

Bronzino, J. D., 2006. *The Biomedical Engineering Handbook.* Third Edition. CRC Press, Taylor & Francis Group.

Burbeck, S., 1992. *Applications Programming in Smalltalk-80: How to use Model–View–Controller (MVC).*

Caprini, J. A., 2005. *Thrombosis risk assessment as a guide to quality patient care.* Dis Mon, vol. 51, no. 2–3, pp. 70–78.

Chen, N., 2006. *Convention over configuration.*

Cohen, A. et al., 2007. *Venous thromboembolism (VTE) in Europe: The number of VTE events and associated morbidity and mortality.* Thromb Haemost, 98(4):756-764.

Cohen, A. T. et al., 2008. *Venous thromboembolism risk and prophylaxis in the acute hospital care setting (ENDORSE study): a multinational cross-sectional study.* The Lancet, 371(9610), 387–394.

Cohen, S., Kamarck, T. & Mermelstein, R., 1983. *Cohen, S., Kamarck, T., & Mermelstein, R.. A global measure of perceived stress.* Journal of Health and Social Behavior, 24, 385-396.

Curado, M. A. S., 2016. *A medida e as escalas de avaliação da saúde das populações neonatais e pediátricas.* Lisboa: Universidade de Lisboa. Tese de doutoramento.

Dargaud, et al., 2009. *A risk score for the management of pregnant women with increased risk of venous thromboembolism: a multicentre prospective study.* British journal of haematology 145 6: 825-35.

Django Framework. *Django makes it easier to build better Web apps more quickly and with less code.* [Online] Available at: https://www.djangoproject.com/ [Acesso em setembro 2018].

Enderle, J. D. & Bronzino, J. D., 2012. *Introduction to Biomedical Engineering.* Elsevier. Third Edition. Academic press series in biomedical engineering. ISBN 978-0-12-374979-6.

Erl, T., 2009. *Tutorial sobre Web Services.* [Online] Available at: https://www.devmedia.com.br/web-services/2873 [Acesso em abril 2018].

Express Framework. *Fast, unopinionated, minimalist web framework for Node.js.* [Online] Available at: https://expressjs.com/ [Acesso em setembro 2018].

Ferrari, J. F. & Dalacorte, R. R., 2007. *Use of Yesavage Geriatric Depression Scale to evaluate the prevalence of depression in inpatient elderly subjects.* Sci Med, 17(1), 3-8.

Ferreira, P., Miguéns, C., Gouveia, J. & Furtado, K., 2007. *Risco de desenvolvimento de úlceras de pressão: Implementação nacional da escala de Braden.* Loures: Lusociência.

França, A. et al., 2011. *A. França, A. Reis, A. Paulino, C. Lohman, D. Cartucho, G. Campello, L. Morais, P. Moreira, R. Abreu, T. Abreu, Venous thromboembolism risk factors and practices of prophylaxis: ENDORSE study results in Portugal.* Acta Med Port, vol. 24, no. 6, pp. 951-960.

FSW-CEULP, 2016. *Funcionamento de uma Aplicação Web.* [Online] Available at: https://fsw-ceulp.gitbooks.io/desenvolvimento-de-software-para-a-web/web/ [Acesso em março 2018].

Grails Framework. *A Powerful Groovy-based Web Application Framework for the JVM Built on Top of Spring Boot.* [Online] Available at: https://Grails.org/ [Acesso em março 2018].

Grégoire, M. C. & Finley, A. G., 2008. *Doctor, I Think my Baby is in Pain: The assessment of infants' pain by health professionals.* Journal of Pediatric, 84,6-8.

Huffines, B. & Lodgson, M. C., 1997. *The Neonatal Skin Risk Assessment Scale for predicting skin breakdown in neonates.* Issues Comprehensive Pediatrics Nursing, 20, 103-114.

Hughes, C. P. et al., 1982. *A new clinical scale for the staging of dementia.* British Journal of Psychiatry, 140, 566-572.

Kahn, S. R. et al., 2012. *Prevention of VTE in Nonsurgical Patients: Antithrombotic Therapy and Prevention of Thrombosis.* 9 ed. American College of Chest Physicians Evidence-Based Clinical Practice Guidelines Chest;141;e195S-e226S.

Laravel Framework. *Love beautiful code? We do too.* [Online] Available at: https://laravel.com/ [Acesso em setembro 2018].

Laryea, J. & Champagne, B., 2013. *Venous thromboembolism prophylaxis.* Clin Colon Rectal Surg, 26(3):153–9.

Manjunath, M., 2018. *AngularJS and Angular 2+: a Detailed Comparison".* [Online] Available at: https://www.sitepoint.com/*Angular*js-vs-*Angular*/ [Acesso em setembro 2018].

MDN Web Docs, 2018. *Cross-Origin Resource Sharing (CORS).* [Online] Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS [Acesso em agosto 2018].

Meteor Framework. *The fastest way to build JavaScript apps.* [Online] Available at: https://www.meteor.com/ [Acesso em setembro 2018].

Nations, D., 2018. *What Exactly Is a Web Application?.* [Online] Available at: https://www.lifewire.com/what-is-a-web-application-3486637 [Acesso em março 2018].

Nelson-Piercy, C., MacCallum, P. & al, e., 2015. *Reducing the Risk of Venous Thromboembolism during Pregnancy and the Puerperium (Green-top Guideline No. 37a).* London: Royal College of Obstetricians and Gynaecologists.

Node JS Framework. *Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.* [Online] Available at: https://nodejs.org/en/ [Acesso em setembro 2018].

Okuhara, A., Navarro, T. P., Procópio, R. & Leite, J., 2015. *Incidence of deep venous thrombosis and stratification of risk groups in a university hospital vascular surgery unit.* Jornal Vascular Brasileiro, 14(2), 139-144.

Opensoft, 2017. *Web service: o que é, como funciona, para que serve?.* [Online] Available at: https://www.opensoft.pt/web-service/ [Acesso em abril 2018].

Palmeira, T., 2012. *Como funcionam as aplicações web.* [Online] Available at: https://www.devmedia.com.br/como-funcionam-as-aplicacoes-web/25888 [Acesso em março 2018].

Pavão, J. et al., 2017. *Application for VTE stratification and risk assessment.* Lisboa: 12th Iberian Conference on Information Systems and Technologies (CISTI), pp 1-6.

Pinho, C. et al., 2013. *Recomendações de Tromboprofilaxia em Cirurgia Ambulatória.* APCA.

Play Framework. *The High Velocity Web Framework For Java and Scala.* [Online] Available at: https://www.playframework.com/ [Acesso em setembro 2018].

Plohmann, O., 2012. *Groovy 2.0 Performance compared to Java.* Samstag.

Prandoni, P. et al., 1997. *The clinical course of deep-vein thrombosis. Prospective long-term follow-up of 528 symptomatic patients.* Haematologica, 82, 423-428.

Pyramid Framework. *Pyramid The Start Small, Finish Big Stay Finished Framework.* [Online]
Available at: https://trypyramid.com/ [Acesso em setembro 2018].

React Framework. *A JavaScript library for building user interfaces.* [Online] Available at: https://reactjs.org/ [Acesso em setembro 2018].

Reenskaug, T. & Coplien, J., 2009. *The DCI Architecture: A New Vision of Object-Oriented Programming.*

Rose, A., 2018. *Venous Thromboembolism Prophylaxis – Adult – Inpatient/Ambulatory – Clinical Practice Guideline.* U. W. Health, University of Wisconsin Hospitals and Clinics.

Ruby on Rails Framework. *Imagine what you could build if you learned Ruby on Rails….* [Online]
Available at: https://rubyonrails.org/
[Acesso em março 2018].

Silva, A., Almeida, G., Cassilhas, R. & al, e., 2007. *Equilíbrio, coordenação e agilidade de idosos submetidos à prática de exercícios físicos resistidos.* Revista Brasileira de Medicina do Esporte, 14 (2), 88-93.

Silva, J. M. R., 2018. *Avaliação do Risco de Diabetes Mellitus Tipo 2 na População Adulta de uma Unidade de Saúde Familiar.* Guarda: Escola Superior de Saúde Instituto Politécnico da Guarda.

Spielberger, C. D., 1973. *Preliminary manual for the State-Trait Anxiety Inventory for Children.* Palo Alto, CA: Consulting Psychologists Press.

SPMS, 2018. *RSE – Registo de Saúde Eletrónico.* [Online]
Available at: http://spms.min-saude.pt/product/area-cidadao/
[Acesso em setembro 2018].

Spring Framework. *Spring: the source for modern Java.* [Online]
Available at: https://spring.io/
[Acesso em setembro 2018].

Stevens, B., Johston, C., Petryshen, P. & Taddio, A., 1996. *Premature infant pain profile: Development and initial validation.* Clinical Journal of Pain, 12(1), 13-22.

Symfony Framework. *Symfony is a set of reusable PHP components... and a PHP framework for web projects.* [Online]
Available at: https://symfony.com/
[Acesso em setembro 2018].

Thoyre, S. M., Shaker, C. S. & Pridham, K. F., 2007. *The Manual of Early Feeding Skills - EFS.* Chapel Hill: The University of North Carolina.

TypeScript. *TypeScript. JavaScript that scales.* [Online] Available at: https://www.Type*Script*lang.org/index.html [Acesso em março 2018].

Van Es, N. et al., 2017. *The Khorana score for the prediction of venous thromboembolism in patients with pancreatic cancer.* Thrombosis Research, 150, 30–32.

Vaz, P. S., Duarte, L. & Paulino, A., 2012. *Risco e Profilaxia do Tromboembolismo Venoso em Doentes Cirúrgicos.* Revista Portuguesa de Cirurgia, n. 23, p. 23-32. ISSN 2183-1165.

Verso, M. et al., 2012. *A modified Khorana risk assessment score for venous thromboembolism in cancer patients receiving chemotherapy: the Protecht score.* Internal and Emergency Medicine, 7(3), 291–292.

VSCHART. *vsChart.com The Comparison Wiki.* [Online] Available at: http://vschart.com/ [Acesso em setembro 2018].

Vue Framework. *The Progressive JavaScript Framework.* [Online] Available at: https://vuejs.org/ [Acesso em setembro 2018].

W3schools.com. *The world's largest web developer site.* [Online] Available at: http://www.W3Schools.com [Acesso em fevereiro 2018].

Wells, P. S., Anderson, D. R., Rodger, M. & al, e., 2001. *Excluding pulmonary embolism at the bedside without diagnostic imaging: management of patients with suspected pulmonary embolism presenting to the emergency department by using a simple clinical model and d-dimer.* Annals of Internal Medicine. 135: 98-107.

Wells, P. S., Anderson, D. R., Rodger, M. & et al., 2003. *Evaluation of D-dimer in the diagnosis of 8 suspected deep-vein thrombosis.* New England Journal of Medicine 349: 1227–35.

Yii Framework. *Yes, it is! Yii is a fast, secure, and efficient PHP framework. Flexible yet pragmatic. Works right out of the box. Has reasonable defaults.* [Online]

Available at: https://www.yiiframework.com/ [Acesso em setembro 2018].

Yoon, J., Davtyan, C. & Schaar, M. v. d., 2017. *Discovery and Clinical Decision Support for Personalized Healthcare.* IEEE Journal of Biomedical and Health Informarics, 21(4), 1133-667.

Zend Framework. *Your Innovation. Delivered Faster.* [Online] Available at: http://www.zend.com/ [Acesso em setembro 2018].