Mestrado em Engenharia Informática
Estágio
Relatório Final

TICE.GenericEntity – Gestor de Entidades e Gerador de Formulários

Pedro Gil Lopes Castelo Branco Catré
catre@student.dei.uc.pt

Orientadores:
Professor Doutor Ernesto Costa

Engenheiro Alcides Marques

Data: 12 de Julho de 2012

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

O TICE.GenericEntity foi desenvolvido no âmbito da iniciativa TICE.Healthy, um projeto a nível nacional no qual se pretende desenvolver novos produtos e serviços para os mercados "Saúde e Qualidade de Vida", catalisando as empresas copromotoras para a criação de serviços que utilizem como suporte a infraestrutura Internet.

Este estágio está inserido na primeira linha de ação deste projeto, a We.Can, que constitui uma plataforma com vertente simultaneamente agregadora e fornecedora de dados, onde está planeada a criação de um registo clínico eletrónico pessoal (*Personal Health Record* – PHR) que será disponibilizado aos outros subprojectos. Uma vez que atores diferentes terão diferentes requisitos para o PHR, que os registos clínicos têm uma forte componente técnica na área de saúde (que geralmente tem que ser refinada ao longo do tempo com a ajuda de profissionais da área) e que existe um grande número de patologias e condições específicas que podem ser contempladas, torna-se aparente a necessidade de permitir a extensão e especificidade do PHR, não só pelos subprojectos do TICE.Healthy, mas também por outros fornecedores de aplicações que venham a juntar-se à plataforma.

Para dar resposta às necessidades foi concebido um módulo, o TICE.GenericEntity, que possibilita não só a configuração do registo clínico eletrónico de forma extremamente flexível e com custo mínimo para o projeto, mas também o desenvolvimento de extensões do mesmo de forma dinâmica. A integração do TICE.GenericEntity num projeto apresenta enormes vantagens, possibilitando a modificação da estrutura de dados sem necessidade de *redeploy* e tornando fácil e rápida a introdução de modificações na interface de visualização e lógica de negócio, com o mínimo de programação. Este componente, sobre o qual está configurado o PHR, é genérico para que possa servir de base a outros projetos de natureza distinta, dotando-os dos benefícios e características referidas e diminuindo os custos necessários ao seu desenvolvimento.

Palavras-Chave

"Base de Dados", "Schema-Free", "Gerador de Formulários", "Gerador de Código", "TICE.Healthy", "Motores de Regras de Negócio"

Agradecimentos

A quem não posso deixar de agradecer:

Aos meus orientadores, Professor Doutor Ernesto Costa e Engenheiro Alcides Marques, pela inspiração, pelo apoio e pela amizade com que me distinguiram em todas as etapas do estágio.

Ao Mestre João Quintas e ao Engenheiro Gouveia Leal pelos incentivos e acompanhamento do trabalho.

Ao Professor Doutor Carlos Bento e ao Professor Doutor Paulo Rupino pelos conselhos pertinentes e críticas construtivas.

Ao Engenheiro Miguel Machado pelo enorme contributo para a minha formação e desafios oportunos. Aos restantes colegas de equipa: Diogo Machado, Nuno Rebelo, David Cardoso, David Francisco, Ricardo Vitorino, Gonçalo Ferrão, Diana Guardado e outros membros do Instituto Pedro Nunes, pela amizade, pelo espírito de companheirismo e por estarem sempre dispostos a ajudar.

Aos meus amigos, com quem pude contar tanto nos bons momentos como nos mais difíceis.

À minha sobrinha Maria por me dar um sorriso sempre que eu mais preciso, à minha irmã Dora pela paciência e dedicação e aos meus pais, Maria Francelina e Alcides Catré, por não deixarem que nada me falte.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

ACID Atomicidade, Consistência, Isolamento e Durabilidade

AHR *Animal Health Record*

AJAX *Asynchronous JavaScript and Extensible Markup Language*

AMD *Asynchronous Module Definition*

API *Application Programming Interface*

BASE *Basically Available, Soft state, Eventual consistency*

CAPTCHA *Completely Automated Public Turing Test To Tell Computers and Humans Apart*

CSV *Comma-Separated Values*

DOM *Document Object Model*

DRY *Don't Repeat Yourself*

EAV Entidade-Atributo-Valor

EHR *Electronic Health Record*

GUI *Globally Unique Identifier*

HIS *Health Information Systems*

HL7 *Health Level 7*

HTML *HyperText Markup Language*

HTTPS *HyperText Transfer Protocol Secure*

IP *Internet Protocol*

IPN Instituto Pedro Nunes

JDBC *Java Database Connectivity*

JSON *JavaScript Object Notation*

LIS Laboratório de Informática e Sistemas

MVCC *Multiversion Concurrency Control*

ORM *Object-Relational Mapping*

OWASP *Open Web Application Security Project*

PHR *Personal Health Record*

REST *REpresentational State Transfer*

RIM *Reference Information Model*

RSS *Really Simple Syndication*

SGBD Sistema de Gestão de Bases de Dados

SGBDR Sistemas de Gestão de Bases de Dados Relacionais

SOAP *Simple Object Access Protocol*

TCP *Transmission Control Protocol*

TICE Tecnologias de Informação Comunicação e Eletrónica

WYSIWYG *What You See Is What You Get*

XML *Extensible Markup Language*

XMPP *Extensible Messaging and Presence Protocol*

Capítulo 1
Introdução

O TICE.Healthy é um projeto de investigação e desenvolvimento suportado por um consórcio de 24 empresas e instituições de Investigação e Desenvolvimento cujo objetivo é conceber produtos inovadores na área do *e-health*. A sua principal finalidade é potenciar a presença de empresas e organizações portuguesas nos mercados globais da área estratégica de "Saúde e Qualidade de Vida", pela criação de um ecossistema de serviços e aplicações onde utentes, familiares e profissionais de saúde podem cooperar em atividades do dia-a-dia. A primeira linha de ação da iniciativa TICE.Healthy é a criação de um canal de informação e interação para venda de produtos e serviços de saúde, que irá providenciar um registo clínico eletrónico pessoal (*Personal Health Record* – PHR) [1].

Este PHR irá suportar outros subprojectos, ou seja não só os atuais, mas também os de fornecedores de serviços que venham a fazer parte da plataforma, pelo que terá que dar resposta às necessidades de diversos parceiros, serviços e aplicações com requisitos evolutivos. Considerando que os registos de saúde num PHR provêm de vários hospitais e organizações, o repositório terá que estar pronto para suportar diferentes formatos de dados e estruturas imprevisíveis [2]. De facto, apesar de os dados clínicos típicos estarem bastante estandardizados, este repositório irá ainda armazenar dados de serviços de cuidados informais que não são uniformes. Para além disso, neste, tal como na maioria dos projetos de engenharia de *software*, existe um grande peso associado a mudanças de requisitos ou adição de novas funcionalidades que tipicamente forçam revisões não só às camadas de acesso a dados e regras de negócio, como também modificações à interface de visualização. O código produzido nestes cenários é repetitivo e o seu desenvolvimento é lento e potencialmente sujeito a muitos erros. Adicionalmente, tais mudanças não seriam diretamente aplicáveis no ambiente de produção, ou refletidas de imediato no *browser*, como seria desejável.

Para dar resposta às necessidades apresentadas, o principal objetivo deste estágio é o desenho e implementação de um módulo, o TICE.GenericEntity, que permite gerir entidades e gerar os componentes para visualizar e manipular os registos dessas entidades de forma dinâmica, mesmo em ambiente de produção. De forma complementar foram concebidos e implementados mecanismos para criar de raiz, e modificar dinamicamente, aplicações de visualização de dados completas com menus/submenus que contêm componentes. A Figura 1 apresenta um exemplo ilustrativo em que todos os elementos podem ser criados, configurados e geridos pelo painel de administração em ambiente de produção.

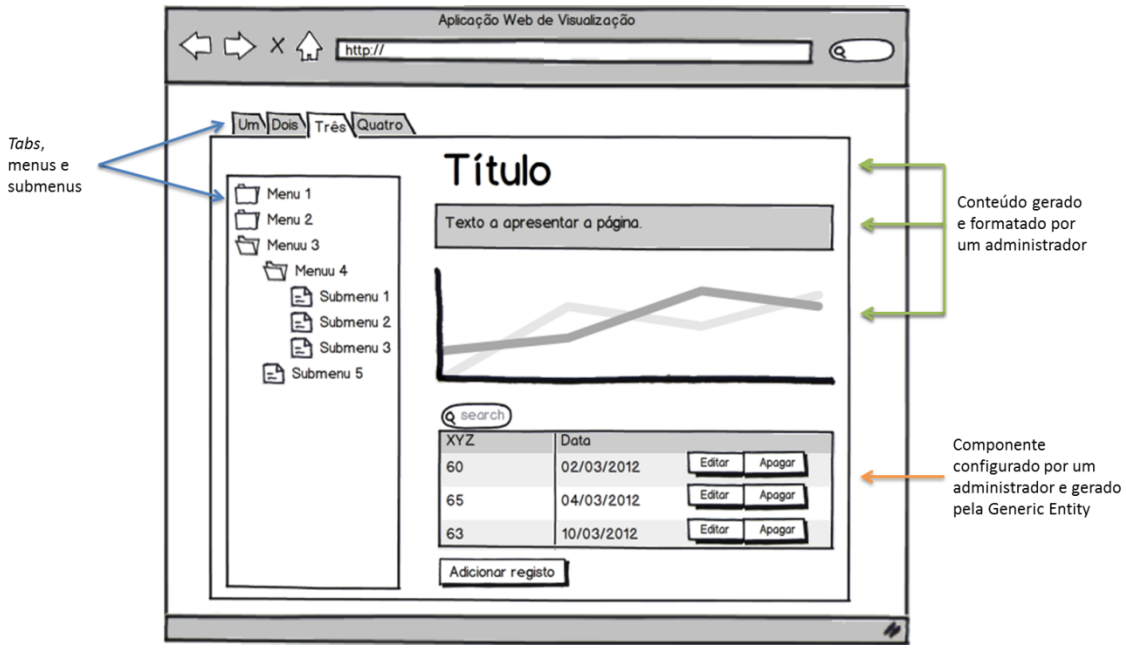

Figura 1 Exemplo ilustrativo de uma aplicação web configurada com a Generic Entity.

Como se pode observar a aplicação de visualização não está vinculada ao âmbito da saúde, podendo ser utilizada noutras áreas de negócio. Este estágio foca-se então na criação do módulo referido e não especificamente no desenvolvimento de um *Personal Health Record*. No entanto, faz parte do âmbito do projeto configurar um PHR suportado pelo TICE.GenericEntity, como exemplo de um caso de uso desta ferramenta. De facto, pretende-se que o TICE.GenericEntity possa ser empregue em contextos e projetos completamente diferentes dos da área de saúde. Afigura-se que é do interesse do Laboratório de Informática e Sistemas do Instituto Pedro Nunes que esta ferramenta seja genérica para que possa ser usada como base para outros projetos, diminuindo o seu tempo de desenvolvimento e permitindo herdar todos os benefícios inerentes a integração deste componente. De facto, a possibilidade de modificar a estrutura de um repositório de dados sem ser necessário *redeploy* e a facilidade e rapidez de criação e modificação das formas de visualizar os dados e lógica de negócio são benefícios altamente desejados para diversos projetos de desenvolvimento *web*.

No caso específico do PHR pretende-se que seja possível desenvolver extensões do mesmo de forma dinâmica. O TICE.GenericEntity apresenta uma API permitindo o seu uso pelos fornecedores de aplicações da plataforma. É ainda disponibilizado um editor, que, complementando o TICE.GenericEntity, permite a um administrador adicionar e configurar extensões do PHR através de controlos *web* visuais. Os campos personalizados que podem ser criados são dinâmicos para que possam ser modificados mesmo em *runtime*. Um exemplo de utilização destas capacidades seria a criação direta, em ambiente de produção, de uma nova vista com componentes *web* customizados para registo e suporte de uma qualquer patologia. Tal permitirá aos fornecedores de aplicações com acesso bloqueado à base de dados e sistemas, uma forma de adequar a estrutura de dados às suas necessidades e ao mesmo tempo alcançar uma poupança no tempo de desenvolvimento de modificações e/ou novas funcionalidades no registo clínico.

Exemplos concretos da aplicabilidade das características de extensão e flexibilidade do PHR a criar são:

- O subprojecto Mind.Care – onde é apresentada a proposta de criação de registos eletrónicos específicos para as doenças de Alzheimer e de Parkinson que irão, em princípio, ser uma extensão do PHR.
- O subprojecto Sensor Care – que, em traços gerais, se foca na monitorização de utentes através de sensores. Estes transmitem a informação para uma aplicação Android que utiliza o TICE.GenericEntity como repositório para armazenamento e consulta de dados e como retransmissor desses mesmos dados, para que estes possam ser visualizados em tempo real utilizando um *browser*.

O PHR configurado neste estágio constitui uma prova de conceito que pretende demonstrar a adaptabilidade do sistema aos objetivos. Foi ainda definido pela direção do projeto que os esforços deste estágio seriam concentrados na vertente funcional da ferramenta. Contudo, na solução desenhada foram tidos em consideração requisitos não funcionais, com especial destaque para as questões de segurança e privacidade, garantindo assim que nada do que foi até agora implementado impede a integração e ativação dos mecanismos de segurança que venham a ser desenvolvidos. Foi feita uma análise das questões de segurança mais importantes a considerar (que é apresentada no documento anexo "TICE-GenericEntity Análise de Segurança").

1.1 Estrutura do Relatório

Feita a introdução, as outras componentes do documento organizam-se da seguinte forma: o capítulo 2 apresenta o planeamento e processos de engenharia; o capítulo 3 expõe a análise do estado da arte realizado; o capítulo 4 descreve a fase de identificação dos requisitos do projeto; no capítulo 5 é descrita a arquitetura da ferramenta e solução desenvolvida; no capítulo 6 são apresentados os testes realizados; no capítulo 7 são sumarizados os resultados do estágio e benefícios do projeto; e no capítulo 8 são expostas as conclusões do trabalho e o rumo a tomar após o término do estágio.

Por fim também foram elaborados documentos, tanto do projeto (para os parceiros e futuros *developers*) como anexos do relatório. Para anexos específicos do relatório foi criado o documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)", que apresenta complementos às várias secções do relatório sempre que tal se justificou.

Capítulo 2
Planeamento

Este capítulo foca-se em questões de gestão de projeto, tais como procedimentos internos, organização e planeamento.

2.1 Marcos do Projeto

De forma semelhante ao Scrum[1], este projeto foi dividido em três grandes fases:

- Fase *pregame*: em que foi feito o planeamento, modelação de negócio, requisitos e desenho da arquitetura do sistema.
- Fase *game*: constituída por cinco *sprints*[2] de desenvolvimentos (ciclo desenho-implementação-teste), cada um com a duração de um mês, que foram inspirados tanto quanto possível nos *sprints* do Scrum.
- Fase *postgame*: em que o processo de desenvolvimento termina e o projeto é preparado para ser lançado. Esta fase incluiria integração, testes finais, escrita de manual de instalação e *deploy* do módulo desenvolvido, mas todas essas tarefas foram contempladas em *sprints* logo a fase *postgame* passou essencialmente a ser a finalização e melhoria de documentação do projeto.

Na Figura 2 e na Figura 3 são apresentados os diagramas de Gantt com o plano concebido e desvios para o primeiro semestre e segundo semestre, respetivamente.

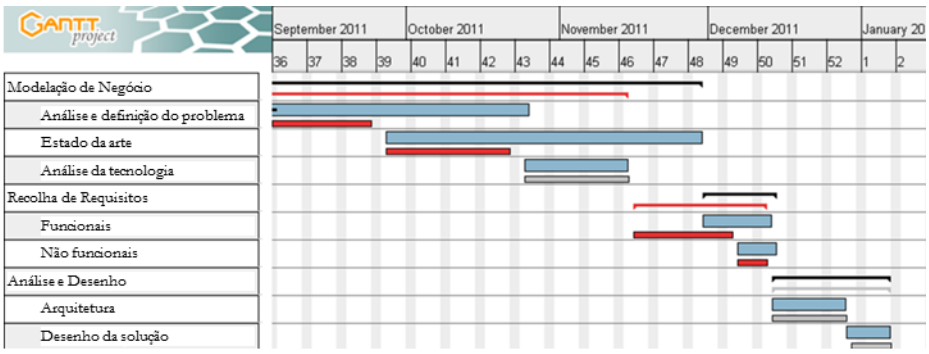

Figura 2 Diagrama de Gantt planeado para o primeiro semestre (fase pregame) e desvios.

A azul está representado o tempo real e a vermelho o planeado.

[1] Scrum é uma *framework* de desenvolvimento ágil.
[2] Neste contexto, de forma semelhante ao Srum, *sprint* é um período durante o qual se concentram esforços para atingir objetivos fixos.

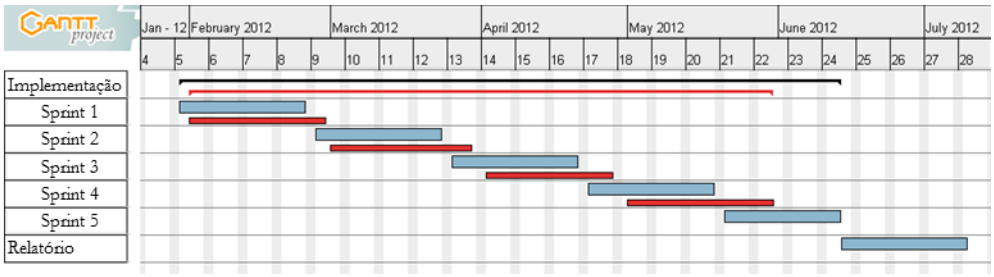

Figura 3 Diagrama de Gantt do segundo semestre (fases game e postgame) e desvios.

A azul está representado o tempo real e a vermelho o planeado.

O planeamento inicialmente feito teve em conta a falta de definição relativamente ao projeto que ia constituir o estágio dentro da iniciativa TICE.Healthy. Consequentemente, foi reservado bastante tempo para a definição e análise do problema. Ainda assim, como se pode observar na Figura 2, ocorreram desvios relativamente ao planeamento.

A definição e análise do problema foram atrasadas por complicações verificadas na comunicação com os parceiros do projeto. Os contornos dos problemas a necessitar de resposta não eram claros e os recursos com os quais se podia contar eram desconhecidos. Especificamente era impossível saber qual o trabalho envolvido na implementação do PHR sem uma reunião prévia com a empresa Health Information Systems, que demorou a responder aos pedidos de marcação e, na primeira reunião, não tinha toda a informação ou a documentação necessárias. Sem estes esclarecimentos não era possível saber se já estaria implementada alguma parte do sistema que pudesse ser fornecida (como por exemplo o modelo de dados) e se haveria algum requisito especial ou *standard* a considerar no seu desenho. Como tal, nesta fase, não eram claros os desafios de engenharia que poderiam ser enfrentados na realização do trabalho, ou a complexidade dos mesmos, sendo questionável a sua adequação como objeto de estágio.

Esta indefinição inicial do problema em causa levou à produção de trabalho proveitoso para o TICE.Healthy, mas que se revelou inútil para o contexto atual do estágio. Um exemplo disto foi a investigação de *indoor positioning systems* para um outro subprojeto do TICE.Healthy, que foi realizada durante o início do mês de Setembro.

Outro exemplo foi a elaboração do estado da arte para a construção de um módulo de permissões para toda a plataforma. Apesar de o módulo de permissões estar ligado ao atual projeto, não revelou questões de engenharia suficientemente complexas para constituir por si só um estágio e, como tal, deveria ser desenvolvido como uma tarefa secundária. Contudo, o grande atraso inicial impossibilitou o desenvolvimento atempado de um módulo completo, robusto e que satisfizesse os novos requisitos levantados pelo Gestor Técnico dos TICE (que se juntou à equipa a meio do mês de Outubro), sem comprometer e arriscar toda a implementação do módulo Generic Entity. Como tal este risco foi precavido cessando de imediato todos os esforços sobre este módulo e aplicando-os à Generic Entity. Entretanto tomámos conhecimento de que, ao contrário do esperado, a HIS não iria poder fornecer qualquer documentação útil ou componentes previamente desenvolvidos adequados à integração neste projeto. Concretamente, era expetável termos acesso a uma especificação do modelo de dados que não pôde ser fornecida.

Também ocorreu uma mudança de um requisito organizacional central do projeto. Numa primeira fase este estágio deveria ser desenvolvido utilizando a *framework* Django e, após algumas reuniões gerais, toda a plataforma passou a ter que apresentar o *backend*

obrigatoriamente programado em Java, havendo liberdade para escolha da *framework web* a utilizar. Esta ocorrência levou à necessidade de uma análise e escolha da *framework* Java a utilizar e de novo investimento de aprendizagem.

O plano para recuperar o atraso passou por um maior esforço e maior número de horas despendidas com o estágio, e por diminuir os esforços em algumas tarefas em prol de tarefas críticas. Por exemplo, a análise das *frameworks* Java não foi tão profunda quanto o desejado, o que não foi ideal, mas se revelou necessário. Ainda assim há uma grande confiança que a *framework* eleita é a mais adequada.

No segundo semestre o primeiro *sprint* começou mais cedo do que o planeado e decidiu-se adicionar mais um *sprint* aos 4 definidos inicialmente. As tarefas de testes finais, manual de instalação e o *deploy* foram absorvidos pelos *sprints*, em vez de se concentrarem na fase *postgame*. A fase final que foi adicionada consiste essencialmente na escrita do relatório e melhorias à documentação anexa e constituem a fase *postgame* deste projeto.

2.2 Processo de Engenharia

Foram seguidas algumas práticas relacionadas com o desenvolvimento do projeto e com a sua gestão. As mais importantes são apresentadas nas subsecções seguintes.

2.2.1 Processo de Desenvolvimento de Software

A metodologia de *software* adotada é baseada em desenvolvimento ágil[3]. Em especial é procurada inspiração no processo ágil Scrum [3], que é dos métodos mais populares e aclamados e que pretende ser simples e leve. Os artefactos resultantes do uso do Scrum são apresentados no documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)" no Apêndice B Artefactos do Scrum.

A razão pela qual se procura inspiração em metodologias de desenvolvimento ágil é que estas são geralmente direcionadas para:

- Melhorar a capacidade de responder rapidamente a mudanças nos requisitos.
- Cortar em tempos de desperdício ou espera.
- Reduzir o *stress* da equipa e ao mesmo tempo aumentar a produtividade.

Estes benefícios são todos altamente desejados no contexto do projeto pela sua natureza dinâmica (os fatores externos mudam e os objetivos de projeto e planeamento têm que se adaptar) e pelas metas que têm de ser cumpridas.

A divisão de papéis na metodologia ágil utilizada foi também apoiada em Scrum, e foi adotada a nomenclatura deste para a apresentar:

- **Product Owner –** Engenheiro Alcides Marques
- **Scrum Master –** Engenheiro Miguel Machado
- **Scrum Team –** Engenheiro Diogo Machado; Engenheiro Ricardo Vitorino; David Cardoso; Pedro Catré; David Francisco; Nuno Rebelo; Gonçalo Ferrão.

[3] O desenvolvimento ágil é uma forma de desenvolver projetos que dá enfase, entre outros fatores, à adaptabilidade, caminhos curtos entre ideia e implementação, e simplifica formas de colaboração. Exemplos de metodologias ágeis incluem o Extreme Programming (XP) e o Scrum.

2.2.2 Reuniões Diárias

A equipa dos TICE tem reuniões diárias de curta duração (5 a 10 minutos) em que os elementos presentes falam brevemente sobre as tarefas que realizaram desde a última reunião, o que irá ser feito até à próxima reunião e os desafios com que se estão a deparar. Os objetivos destas reuniões são a identificação e remoção rápida dos obstáculos ao trabalho e que toda a equipa tenha uma visão geral do progresso do projeto. De facto, qualquer membro da equipa (estagiário ou não) tem uma visão e compreensão global de toda a plataforma e não apenas do projeto específico que se encontra a desenvolver. Estas reuniões são baseadas nos Daily Scrums, as reuniões diárias da metodologia Scrum.

2.2.3 Reuniões Mensais

Ocorrem ainda reuniões no final de cada mês para apresentar o que foi feito e os objetivos para o mês seguinte. Estas reuniões permitem uma reflexão sobre o que já foi feito, se os objetivos estão a ser cumpridos e que tarefas são prioritárias para o mês seguinte.

2.2.4 Git-Flow

Na implementação é usado o sistema de controlo de versões Git [4], mais especificamente é empregue um conjunto de extensões, Git-flow [5], que providenciam operações de alto nível sobre o repositório seguindo o modelo de *branching* de Vincent Driessen [6].

Para além dos *branches master*, com código pronto para produção, e *develop*, com a versão mais recente das modificações adicionadas para o próximo lançamento, este modelo de desenvolvimento usa uma variedade de *branches* de suporte (*branches*: *Feature*, *Realease* e *Hotfix*) para ajudar no desenvolvimento paralelo entre membros de equipa, facilitar a monitorização de novas funcionalidades, preparar para lançamentos de produção e assistir à rápida resolução de problemas na versão de produção [6], [7]. Os benefícios de trabalho em equipa não foram aproveitados uma vez que toda a implementação foi feita pelo estagiário, mas já há planos para que isso mude no fim do estágio.

2.2.5 Análise de Riscos

A gestão de risco num projeto deve ser contínua para ser eficaz e, como tal, durante o estágio foi mantida uma lista de riscos consultada com frequência. Alguns riscos inicialmente identificados tornaram-se obsoletos com a reestruturação do projeto e deixaram de ser vigiados. Para alguns riscos, as ações de mitigação a tomar não eram claras e como tal foram fundamentalmente monitorizados. Os principais riscos identificados e sobre os quais se aplicaram ações concretas são apresentados nas tabelas abaixo.

Tabela 1 Risco 01.

Título Atraso do trabalho principal do estágio pelo módulo de permissões		
Tipo Planeamento	**Impacto** Elevado	**Probabilidade** Elevada
Descrição	O módulo de permissões não é o núcleo do estágio e ao estar planeado para ser desenvolvido em primeiro lugar pode causar atrasos que se propagam para o TICE.GenericEntity. Adicionalmente surgiram requisitos de integração deste módulo com várias tecnologias como *Extensible Messaging and Presence Protocol* (XMPP) e sistemas de mensagens *publish-subscribe* que necessitam de considerável estudo prévio.	
Mitigação	Cessar de imediato todos os esforços sobre o módulo de permissões e aplicá-los à Generic Entity.	

Tabela 2 Risco 02.

Título *Deploy* automatizado em ambiente de produção		
Tipo Técnico	**Impacto** Médio	**Probabilidade** Média
Descrição	O estagiário não tem experiência com *deploy* de produção de um projeto *web*, dificultando necessidades específicas deste projeto como automatização do processo tendo em conta questões de segurança. Adicionalmente, também não existem outros elementos do LIS do IPN com essa experiência (nestes requisitos) utilizando a *framework* Play.	
Mitigação	Fazer *deploy* de teste após criação de protótipo, em vez de manter um único *deploy* real apenas no final do estágio.	

Tabela 3 Risco 03.

Título *Deploy* em máquinas críticas		
Tipo Técnico	**Impacto** Elevado	**Probabilidade** Médio
Descrição	O *deploy* será realizado numa máquina remota e há o perigo de que a aplicação de configurações erradas na mesma afete outros sistemas a executar em produção.	
Mitigação	Testar o *script* de *deploy* automático num ambiente virtual representativo na máquina local.	

2.3 Conclusão

Durante o primeiro semestre o planeamento do projeto revelou-se extremamente complicado. Existiram várias incertezas em questões críticas que demoraram muito tempo a ser esclarecidas e que, em alguns casos, impediram o avanço do trabalho ou levaram mesmo à sua reformulação.

Felizmente, os obstáculos mais impeditivos foram ultrapassados e a cada dia houve uma recuperação do atraso inicial com o apoio das práticas apresentadas neste capítulo. O segundo semestre já contava com uma base bem definida sendo a principal questão do planeamento a priorização das tarefas por parte do *Product Owner*, em especial das tarefas de integração com outros sistemas que sofreram algumas alterações (algumas tarefas foram removidas e outras adicionadas). Adicionalmente, como a metodologia de desenvolvimento Scrum está a ser estreada pela equipa dos TICE, os processos adotados foram sendo refinados ao longo dos *sprints*, especialmente no que toca à forma mais correta de introduzir as tarefas, tempo despendido e tempo estimado no sistema de gestão, para se conseguirem

métricas adequadas. Uma das mudanças mais notórias ao processo é que, após o final dos estágios atuais, os *sprints* passam de quatro semanas para duas, de forma a tentar obter da equipa estimativas mais próximas da realidade e uma maior flexibilidade para responder a novas necessidades dos parceiros sem comprometer os *sprints*.

Capítulo 3
Estado da Arte

Neste capítulo é apresentada a investigação e análise que dão suporte e motivam o projeto TICE.GenericEntity.

Em primeiro lugar é apresentado e clarificado o conceito de *Personal Health Record* e as motivações que levaram ao seu aparecimento. Em segundo são examinados alguns PHRs já existentes. De seguida é exposto o estudo feito ao *standard* de interoperabilidade Health Level 7 v3 e como este se poderá encaixar no TICE.Healthy e, mais especificamente, com a Generic Entity. Na secção seguinte são tecidas algumas considerações sobre o formato de comunicação a utilizar entre *Web Services*. São ainda investigadas soluções alternativas de armazenamento persistente. Por fim são apresentadas as motivações para a integração de um motor de regras na Generic Entity e qual o motor de regras escolhido.

3.1 Repositório de Informação Clínica

Apesar do módulo a criar ser genérico e configurável é preciso analisar a sua primeira prova de conceito (o PHR) para melhor compreender e identificar os requisitos. Nesta secção é feita uma análise ao conceito de PHR e a sistemas representativos presentes no mercado.

3.1.1 Personal Health Record

Nos últimos anos foram propostas várias definições diferentes de PHR por múltiplas organizações, mas a maioria das definições coincide ao apresentar o PHR como uma aplicação computorizada que guarda a informação pessoal de saúde de um indivíduo. Com sistemas PHR baseados na *web* é possível a pacientes e profissionais de saúde aceder ao registo clínico de forma segura e independente do local onde o utente procura cuidados médicos [8]. Para além disso é mais fácil a troca de informação e integração com outros sistemas [9].

Contrastando com os registos clínicos tradicionais que se destinam principalmente ao registo de resultados de consultas e exames dentro de uma instituição de saúde, os sistemas PHR são uma inovação disruptiva que inverte o fluxo normal de informação de saúde [10]. Clarificando este ponto, uma vez que ocasionalmente ocorre alguma confusão entre os termos *Personal Health Record* e *Electronic Health Record* (EHR), é importante apontar as diferenças entre estes dois sistemas. Enquanto os sistemas EHR existem para dar respostas às necessidades de informação dos profissionais de saúde, os sistemas PHR reúnem os dados de saúde introduzidos por indivíduos, providenciam informação relacionada com o cuidado do próprio e incluem ferramentas para ajudar o indivíduo a tomar um papel ativo na sua própria saúde. Em parte, os PHRs representam um repositório para dados do utente, mas podem também incluir capacidades de suporte a decisão para auxiliar o paciente e prestadores de cuidados na gestão de condições crónicas [9].

Assim, os sistemas PHR não se destinam a substituir os EHR, mas sim a complementá-los, dando suporte ao paciente na gestão da sua própria informação de saúde [11].

3.1.2 Sistemas Personal Health Record

Nesta secção serão apresentados alguns sistemas PHR representativos das soluções mais populares oferecidas pelo mercado nacional e internacional.

3.1.2.1 Meu Sapo Saúde

A Portugal Telecom (PTC) disponibiliza a título gratuito aos seus Clientes uma solução de ficha clínica eletrónica pessoal, "Meu SAPO Saúde", acessível através do Portal SAPO. O Meu SAPO Saúde é um sistema personalizado de informação que permite registar, organizar e gerir a informação de saúde de um indivíduo e da sua família [12].

Esta solução tem alguns aspetos interessantes, nomeadamente é possível definir metas de saúde e monitorizar a sua evolução e disponibiliza ao utente acesso a um grande conjunto de informação de saúde, dados e conhecimento credíveis. Contudo a utilização deste serviço revelou diversas limitações:

- O serviço apresenta enorme latência. Ficou claro que mesmo um utente com necessidades críticas de registar a evolução da sua patologia, ou de um familiar ou amigo, não se sentiria persuadido a usar este serviço por ser muito lento.
- Dá erros frequentes, em especial de operações que não podem ser completadas com sucesso, sem explicação da razão e nas mais diversas funcionalidades (desde inserir um novo registo a aceder a determinada página). Existem funcionalidades disponíveis que, por ventura, poderão nem estar implementadas uma vez que não chegaram a funcionar nas tentativas realizadas (por exemplo exportação dos dados do PHR e acesso ao PHR de outro utilizador que para tal tenha dado permissão).
- Ao fim de 15 minutos a sessão fecha, quer tenha ou não havido atividade nesse período, o que torna o uso do serviço inconveniente.
- Existem poucas funcionalidades de integração com outras entidades.
- Só existe um perfil de utilizador, o de utente.
- O sistema de permissões tem granularidade alta. As permissões só vão ao nível da entidade e não de cada campo o que pode ser limitativo se, por exemplo, esta tiver um campo de observações que só devesse estar acessível a profissionais de saúde.

3.1.2.2 Google Health

O Google Health [13] foi um serviço de saúde que permitia organizar, monitorizar e atuar sobre a informação de saúde. Através de uma conta Google Health era possível armazenar, gerir e partilhar toda a informação pessoal de saúde e bem-estar num sistema central.

A Google descreveu o seu produto como um PHR, mas também como um modelo um pouco diferente, que, para além de oferecer um local para guardar, gerir e partilhar a informação pessoal de saúde de uma pessoa, também providenciava um diretório de serviços *online* para atuar sobre esta informação diariamente [14]. A estratégia desta plataforma permitia aos doentes importar os seus registos, historial de prescrições e resultados de testes, interagir com serviços e ferramentas como agendamento de consultas e ferramentas de bem-estar à medida que estas eram adicionadas à plataforma por fornecedores de aplicações. O Google Health era baseado em *standards* abertos (*Continuity of Care Record* [15], para troca de dados, e *Simple Object Access Protocol* – SOAP – para a interoperabilidade dos *Web Services*), providenciava uma API de desenvolvimento, bibliotecas de programação e infraestrutura de testes. O Google Health era orientado para os Estados Unidos da América já que os serviços de terceiros não estão disponíveis fora deste país [11].

Importa salientar que o Google Health foi descontinuado a 1 de Janeiro de 2012. Este insucesso acautela que a construção de um PHR não é um processo fácil. Da utilização deste serviço observou-se que este se concentrava em especial em gerir e armazenar informação de saúde *online*. Ao contrário do Sapo Saúde, o Google Health não se apresentava como uma fonte de informação de saúde e bem-estar, as funcionalidades apresentadas funcionaram sempre de acordo com o esperado (sem erros ou grande latência) e tinha alguma integração com entidades exteriores (mas não para Portugal). O sistema de partilha de dados e permissões do Google Health tinha uma granularidade incomportável – muito superior ao Sapo Saúde – para partilhar os dados com outro utilizador tinha de se dar acesso a toda a informação sem qualquer seletividade. Adicionalmente, tal como o Sapo Saúde, o Google Health só tinha um tipo de perfil, o de utente.

3.1.2.3 Microsoft Health Vault

O Microsoft Health Vault é uma plataforma de partilha de dados médicos [16] lançada a 4 de Outubro de 2007, direcionada tanto a utentes como a profissionais de saúde. O Health Vault consiste em dois produtos distintos e gratuitos para os utilizadores: um repositório eletrónico de dados de saúde e motor de busca especializado para informação de saúde [17]. O Health Vault destaca-se do Sapo Saúde, Google Health e outros PHRs por possuir uma rede de parceiros extensa, particularmente na área de dispositivos médicos (como sensores) e equipamento de *fitness*. O Microsoft Health Vault coopera com hospitais, profissionais de saúde e farmácias, mas apenas nos Estados Unidos da América.

Ao contrário do Sapo Saúde, o Microsoft Health Vault constitui-se como uma plataforma com APIs abertas que oferecem infraestrutura técnica para a construção de aplicações por fornecedores de serviços.

3.1.2.4 Considerações Relativamente aos PHRs Analisados

Como já foi referido o Sapo Saúde tem falhas que tornam este serviço pouco atraente. Apesar de mais bem conseguidos, o Google Health e Health Vault têm também limitações graves claramente originadas da desconsideração destes produtos pelas empresas que os criaram. Algumas razões que poderão ter levado à descontinuação do Google Health e ao pouco sucesso do Sapo Saúde e, em menor escala, do Health Vault, e como podem ser respondidos pelo TICE.Healthy, são analisadas de seguida [18], [19]:

- O Google Health e o Sapo Saúde não têm uma componente social ou de diversão. O projeto TICE.Healthy contempla a criação de uma rede social que poderá tornar toda a experiência mais interessante e poderá encorajar os utilizadores a darem maior uso aos serviços disponibilizados. Também existem projetos que se concentram em cuidados informais e que promovem a diversão e o desafio.
- Outra crítica que se pode apontar é que a maioria das pessoas simplesmente não tem tempo ou motivação para manter os seus dados de saúde atualizados. Existe apenas um nicho (especialmente em pessoas com doenças crónicas) para as quais existe um benefício real e até a necessidade de inserir e atualizar os dados de saúde. Para todas as outras pessoas ou este processo é tão automatizado quanto possível ou o PHR não é utilizado. Desta forma, o ecossistema de serviços à volta da plataforma deve automatizar a aquisição e armazenamento de dados relevantes e o PHR deverá ser utilizado primariamente para visualizar e partilhar os dados. Ao ser uma plataforma de armazenamento e partilha de dados médicos o Microsoft Health Vault aplica, em parte, esta estratégia.
- O Google Health e o Sapo Saúde não envolvem profissionais de saúde e estes são essenciais para a adoção generalizada do PHR. Ajudar os utentes a tomar um papel ativo na

sua saúde é importante, mas para a maioria têm de existir benefícios reais decorrentes do uso do PHR. No TICE.Healthy existem serviços planeados para integrar no PHR nesse sentido como a possibilidade de fazer marcação de consultas, renovar o receituário, solicitar documentos (por exemplo: declarações e atestados), inserir o utente em programas de saúde específicos (por exemplo para futuras mães terem acesso a informação e serviços personalizados), interação com o profissional de saúde por mensagens de texto e/ou vídeo, entre outros. Torna-se importante que a Generic Entity permita adicionar facilmente novos componentes (sejam estes programados no projeto ou adicionados de fontes externas). Para além disso está planeado que o PHR seja ligado a fontes de dados fidedignas e confiáveis, nomeadamente a processos clínicos eletrónicos de instituições de acompanhamento de cuidados formais o que tem imenso potencial para aumentar os benefícios para os utentes.

- Os três PHRs analisados tiveram um *marketing* extremamente pobre e muito pouco suporte. Pegando no caso do Google Health, não é surpreendente que este tenha sido descontinuado quando se considera que é necessário muito tempo, dinheiro e parceiros para transformar a indústria, a área da saúde é difícil e não é uma das competências nucleares da Google. Também há que ter em conta que as métricas para avaliar a adoção e sucesso de um projeto para a Google ou Microsoft têm dimensões de ordens de grandeza superiores à da maioria das empresas. Adicionalmente, numa área tão sensível como esta, o modelo de negócio não é simples – por exemplo, o potencial para publicidade é baixo já que pode comprometer a confiança do consumidor. De facto, para as soluções analisadas para além da clara falta de investimento ou suporte parece haver um fraco (ou mesmo inexistente) modelo de negócios.
- Os três PHRs apresentados podem ser criticados por falta de especificidade e completude dos seus formulários para monitorização, o que os pode tornar inúteis em alguns casos. Por exemplo o formulário de inserção de resultados de testes do Google Health era completamente geral tornando-se inútil para resultados de procedimentos mais complexos. Ao aplicar esta crítica ao Health Vault fala-se de serviços do seu ecossistema criados por terceiros – por exemplo um serviço de monitorização de glicemia que não pergunta se os valores são em jejum ou depois de uma refeição (tornando inúteis os valores registados). Este tipo de erros é facilmente explicável se assumirmos que a maioria das aplicações possa ter sido desenhada quase exclusivamente por pessoas com formação apenas em tecnologias de informação e pouco sensibilizadas para as reais necessidades de um utente ou profissional de saúde. Ao ser facilmente configurável, a Generic Entity pode contribuir para que haja um maior foco no desenho das entidades de saúde, ou para que se possam corrigir rapidamente estes erros quando forem detetados.
- Ao não tentar ser uma central de saúde base pronta para utilização, mas simplesmente um repositório para partilha de dados, o Health Vault torna-se mais difícil de começar a utilizar. O seu ecossistema rico em aplicações pode ser uma mais-valia mas é também uma fonte de confusão para os utilizadores e geralmente é necessário escolher mais do que um serviço para ter as funcionalidades base normalmente fornecidas por um PHR (salientando-se ainda que nem todos os serviços são gratuitos). O TICE.Healthy é diferente do Health Vault neste aspeto, pois irá fornecer gratuitamente a sua própria central de saúde com funcionalidades base logo à partida aos utilizadores que se registarem no portal.

Através desta breve análise é possível observar que existem diversos fatores que

podem influenciar o sucesso do TICE.Healthy. Adicionalmente, as propostas de valor e agentes diferenciadores e inovadores dependem de vários projetos em desenvolvimento. O enquadramento e contributo da Generic Entity neste grande consórcio irão passar essencialmente por permitir a fácil configuração e flexibilidade tanto do repositório de dados como da sua visualização, para que possa suportar os outros produtos a serem criados por uma fração do custo normal e promovendo trabalho de maior qualidade. O ecossistema de profissionais de saúde, *software*, aplicações e dispositivos com os quais os utentes se relacionam irão poder partilhar um espaço comum no PHR, trocando informações com o repositório. Assim, os benefícios e potencial da solução concebida são reais, mas para que se concretizem têm que ser aproveitados pelo consórcio.

3.2 Abordagens para Extensibilidade em Runtime

Nesta secção é realizada uma análise a diferentes abordagens para alcançar extensibilidade em *runtime* de funcionalidades e/ou estrutura de dados que modela a interface, de forma semelhante ao pretendido para o TICE.GenericEntity. O objetivo é obter informação sobre estas soluções que auxilie, tanto na construção deste módulo, como na identificação de características e funcionalidades relevantes para implementar no sistema.

Para este efeito foram analisados diferentes produtos. Contudo, uma vez que muitos deles são comerciais e/ou aplicações *web*, e como tal geralmente fechados, nem sempre foi possível estudar a solução implementada. Esses casos serviram apenas de suporte à definição de requisitos funcionais do sistema. A análise detalhada dos produtos estudados encontra-se no ficheiro anexo intitulado "TICE-GenericEntity Estado da Arte".

3.2.1 Produtos Desenvolvidos para Colaboração e Gestão de Conteúdo

Como seria previsível, os sistemas que mais lidam com os requisitos de extensibilidade pretendidos são sistemas de colaboração e gestão de conteúdo. Mesmo quando estas funcionalidades não são suportadas pelo sistema base, geralmente existem extensões para satisfazer a procura.

As características mais relevantes dos sistemas foram analisadas com o objetivo de identificar as funcionalidades comuns que suportem a recolha de requisitos. O estudo realizado não tinha como objetivo a avaliação qualitativa das ferramentas, mas sim a comparação quantitativa das funcionalidades de forma a criar uma primeira base de requisitos para a solução a desenvolver. As características identificadas foram:

- Construção de formulários sem ser necessária qualquer experiência de programação.
- Exportar informação em múltiplos formatos.
- Proteção contra *spambots* via imagens embutidas *Completely Automated Public Turing Test To Tell Computers and Humans Apart*, (CAPTCHA).
- Validação de informação com regras de validação embutidas.
- Suporte para armazenamento em base de dados.
- Dividir formulários em múltiplas páginas.
- Lógica condicional modificando o formulário consoante as escolhas dos utilizadores.
- Possibilidade de receção de notificação de *email* das respostas.
- Suporte para formulários multilíngues.
- Gestão de conteúdo nas empresas (gestão de fluxo de documento e outro conteúdo).
- Procura através de um motor de pesquisa.

- Análises e relatórios.
- Queries *ad-hoc*.
- Controlo de versões de formulários.
- Gestão de vistas.
- Campo de uma entidade para pesquisar/listar informação de outra entidade.
- Marcar entidades como eliminadas (mantendo a sua informação).

A tabela de sistematização destas características encontra-se em anexo no documento "TICE-GenericEntity Estado da Arte" e permitiu observar que o Microsoft SharePoint e Adobe Business Catalyst são as ferramentas mais completas.

A análise realizada constituiu um ponto de partida para a realização deste projeto que se pretendia completo e inovador, possuindo não só as características mais comuns mas também características únicas como a possibilidade de adicionar ou alterar regras de negócio em ambiente de produção e configurar menus e componentes de uma página, com uma forte componente do lado do cliente, para que a interface dinâmica tenha pouca latência.

3.2.2 Abordagens para Armazenamento de Configurações

Em qualquer sistema que pretenda o tipo de extensibilidade descrito são necessários mecanismos para guardar as configurações das entidades que vão sendo criadas. Através da análise de diferentes produtos foram identificadas duas abordagens para alcançar este objetivo, nomeadamente:

1. Guardar toda a definição da entidade e configurações num campo de uma tabela da base de dados num formato de texto, como por exemplo: *Extensible Markup Language* (XML) ou *JavaScript Object Notation* (JSON). Esta é a abordagem seguida pelo Microsoft SharePoint [20].
2. Guardar as configurações em campos de uma ou mais tabelas seguindo um esquema relacional. Esta é a abordagem implementada pelos *plugins* de Wordpress Visual Form Builder [21] e FormBuilder [22], [23].

Guardar toda a definição em XML ou JSON é claramente a abordagem mais vantajosa, uma vez que:

- Se torna possível aplicar serialização automática.
- A manutenção de versões das entidades é muito mais simples.
- Obtém-se um desempenho superior ao reduzir o número de acessos à base de dados, uma vez que num único acesso se obtém toda a definição da entidade. Com efeito, tipicamente esta informação é utilizada de forma atómica e não é comum ser necessário aceder apenas a uma parte dela.

A segunda opção é anómala na medida em que é viável executar *queries* sobre a estrutura das entidades definidas, mas não existe qualquer vantagem derivada desta possibilidade.

3.2.3 Abordagens de Armazenamento de Dados

Nesta subsecção é realizada uma análise a 4 abordagens para armazenamento de dados no repositório e é apresentada a estratégia adotada para implementação na Generic Entity. O objetivo é ter um repositório flexível que lida com novos tipos de dados e atributos sem ser

necessário mudar o esquema físico da base de dados, ao mesmo tempo que continua a suportar a construção fácil de *queries ad-hoc* eficientes.

3.2.3.1 Armazenar Dados Num Formato de Texto

Uma abordagem clássica é o uso de um formato de texto como XML ou JSON para armazenar as chaves (atributos da entidade) e valores correspondentes [2]. Por exemplo, se considerarmos um formulário para preencher como sendo a entidade, os campos (atributos) e respetivas respostas (preenchimento dos campos) seriam armazenados juntos num único campo de uma tabela relacional em XML ou JSON.

Esta solução é fácil de implementar e permite alta extensibilidade. Com efeito, é adequada para casos onde os dados da entidade são manipulados apenas como um todo e de uma forma isolada. A principal desvantagem desta abordagem é que é invulgar aplicar operações diretas ou aceder a atributos específicas usando uma *query* em *Structured Query Language* (SQL) sobre XML ou JSON. Sistemas de gestão de bases de dados relacionais que suportam SQL sobre campos XML apresentam uma degradação de desempenho neste tipo de operações.

3.2.3.2 Entidade-Atributo-Valor

O modelo Entidade-Atributo-Valor (EAV) [24] é empregue frequentemente em cenários clínicos e em outros casos em que o número de atributos, propriedades ou parâmetros que podem ser usados para identificar uma entidade é potencialmente ilimitado. Um desenho EAV tipicamente envolve uma tabela central com 3 colunas que contêm dados que se referem: à entidade, ao atributo e ao valor desse atributo. Uma variante desta solução apresenta várias colunas de diferentes tipos para armazenar o valor, para que seja guardado no tipo de dados correto.

Um exemplo do uso desta técnica seria manter toda a informação sobre um paciente nesta tabela com: coluna chave a identificar o utente (entidade), coluna chave a identificar o atributo e uma coluna para guardar o valor correspondente. Neste caso, uma única instância de uma entidade estaria armazenada ao longo de vários registos da tabela, ao contrário do habitual em que uma instância é armazenada na totalidade num único registo. Adicionalmente, os sistemas de gestão de bases de dados não suportam esta técnica internamente o que prejudica o optimizador de *queries* na seleção de bons planos de acesso. Consequentemente, embora continue a ser possível construir *queries ad-hoc*, essa tarefa não é simples e necessita de operações transformação/*pivot*[4] que irão levar a mau desempenho.

3.2.3.3 Esquema Flexível

Esta abordagem envolve a criação dinâmica de uma tabela com campos específicos por cada nova entidade que seja necessário definir. Por outras palavras, seria utilizada a operação *Create Table* com as colunas necessárias por cada nova entidade.

A principal desvantagem desta abordagem é que a maioria das modificações à entidade levariam a operações *Alter Table* que podem ser lentas se a tabela contiver muitos dados e são, tipicamente, operações bloqueantes.

3.2.3.4 Tabela Genérica com Colunas Pré-Criadas

Esta solução envolve a utilização de uma tabela suficientemente genérica com colunas pré-criadas de vários tipos. Utilizando esta técnica todos os registos correspondentes a cada

[4] Um operador *pivot* comum recolhe os dados em linhas separadas, agrega-os e converte-os em colunas [84].

entidade podem ficar guardados na mesma tabela. Uma ilustração desta abordagem é apresentada na Figura 4.

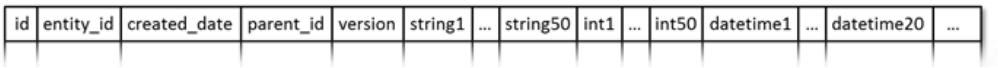

Figura 4 Ilustração da abordagem tabela genérica.

Usando esta técnica todos os registos correspondentes a cada entidade podem ser armazenados na mesma tabela. A plataforma Microsoft SharePoint utiliza uma abordagem semelhante [25]. O uso de tabela genérica revela-se bastante interessante e flexível, continuando a ser possível realizar *queries ad-hoc* sobre os dados. Contudo, a extensão de campos de uma entidade poderá estar limitada ao número de colunas suportado pelo sistema de gestão de bases de dados. Soluções para ultrapassar esta limitação, como por exemplo permitir mais do que um registo por entidade, podem revelar-se bastante complicadas, podem prejudicar o desempenho e tornar as *queries* mais complexas. Ainda assim, um sistema de gestão de bases de dados típico permite o uso de centenas de colunas por tabela, que é considerado mais do que suficiente para representar qualquer tipo de entidade clínica (seja formal ou informal). É importante referir que em bases de dados relacionais esta solução vai originar colunas esparsamente populadas com muitos valores a NULL, o que não é elegante e, normalmente, leva a um desperdício de espaço.

3.2.3.5 Abordagem Híbrida

No TICE.GenericEntity é implementado um híbrido entre tabela genérica e esquema flexível. Desta forma, existe uma tabela genérica com um número razoável de colunas pré-criadas para cada tipo de dados. Se uma entidade precisar de mais campos de um dado tipo são adicionadas colunas de forma dinâmica através de operações *Alter Table* (que devem ser raras e evitadas na medida do possível). Ao contrário do Microsoft SharePoint, este sistema também permite ao utilizador escolher se a nova entidade criada vai usar a tabela genérica do sistema ou uma tabela personalizada especificamente definida. Assim, a solução TICE.GenericEntity, para além de poder criar de raiz um conjunto de entidades sem ser necessário definir um esquema de dados, pode ainda ser utilizada sobre o esquema relacional normal de um sistema (bastando para isso adicionar 3 atributos específicos do sistema a cada tabela) tornando o modelo em causa flexível e permitindo criar interfaces de manipulação e visualização muito rapidamente, tudo isto podendo evitar qualquer desperdício das colunas esparsamente populadas da tabela genérica.

3.3 Health Level 7 v3

O principal objetivo da organização Health Level 7 (HL7) [26] é providenciar *standards* para a troca de dados entre aplicações informáticas de saúde. Ao contrário da versão 2 do HL7, a versão 3 é baseada num modelo de dados orientado a objetos que é denominado *Reference Information Model* (RIM). O RIM contém as classes e respetivos atributos que cobrem o domínio médico e é composto por seis classes base: Act, Participation, Role, Entitity, RoleLink e ActRelationship. No HL7 v3 RIM todos os documentos no domínio da saúde são representados pela classe Act. A classe Participation define o contexto para um Act definindo a relação entre as classes Act e Role. Entidades físicas e seres que participam nos cuidados de saúde são representados pela classe Entity. A classe Role estabelece os papéis das entidades na sua participação em atos de saúde. A classe ActRelationship define a relação entre duas instâncias da classe Act. De forma semelhante, o RoleLink define a relação entre duas instâncias da classe Role [27].

O RIM foi introduzido para harmonizar a definição de mensagens HL7 entre diferentes domínios de aplicação. O HL7 v2 foi extremamente bem-sucedido e, de acordo com a informação que pude obter de algumas reuniões internas, é a versão do *standard* de comunicação mais usado pelas instituições e empresas Portuguesas de saúde. A versão HL7 v3 tem recebido bastante atenção a nível mundial. Contudo, tem sido também sujeita a críticas que abordam questões importantes da sua usabilidade em domínios especializados. As classes base e atributos definidos no RIM são normativos; quando o RIM é aplicado a um novo domínio é necessário selecionar e programar estes atributos [28].

É importante notar que o objetivo do HL7 (inclusivamente do HL7 v3) é a interoperabilidade e tal não obriga, ou prevê, que as aplicações ou bases de dados sejam modeladas com base no RIM [29]. Com efeito, a situação mais comum é que o repositório clínico esteja modelado com uma técnica própria, como por exemplo uma variante da *Entity-Attribute-Value*, e seja usado um *broker* para fazer a tradução, e porventura persistência, das mensagens trocadas utilizando HL7. As instituições Portuguesas utilizam HL7 v2, logo não fazem uso do RIM. Outro exemplo está no artigo publicado em 2011, *Performance Evaluation of Various Storage Formats for Clinical Data Repositories* [30], onde é descrito um sistema que utiliza uma variante de EAV no seu repositório de dados clínicos e utiliza o mirth [31], um motor de integração de cuidados de saúde *open-source* especificamente desenhado para inclusão de mensagens HL7, para filtrar dados e extrair informação relevante para ser armazenada no repositório de dados clínicos. Salienta-se no entanto que, por acaso, surgiram instituições que decidiram implementar aplicações e bases de dados baseadas no HL7 RIM, sem este ser um objetivo ou sequer ter sido previsto pela organização HL7. De facto, considerar o RIM como um candidato válido para os objetos de negócio ou base de dados pode fazer sentido se for previsto trocar múltiplas mensagens no formato HL7 v3, como é o caso para os fornecedores na Holanda onde a infraestrutura de tecnologias de informação do serviço de saúde nacional é baseada no HL7 v3 [29]. Contudo, neste caso não parece haver grande vantagem uma vez que em Portugal se trocam maioritariamente mensagens no formato HL7 v2. Adicionalmente, não é suposto trocar mensagens no formato HL7 entre componentes da mesma plataforma (estas mensagens são para interoperabilidade e têm uma sobrecarga associada que só faz sentido suportar entre sistemas diferentes e não internamente) [32].

Independentemente desta visão torna-se relevante salientar que é possível aplicar a solução Generic Entity sobre uma base de dados modelada pelo RIM, uma vez que qualquer tabela pode ser utilizada e configurada com o módulo desenvolvido. No RIM, as 6 tabelas base são estendidas por centenas de outras tabelas que podem ser configuradas para usar a Generic Entity.

Uma análise mais profunda desta questão, ou uma tentativa de criar uma base de dados baseada em RIM, está fora do âmbito deste estágio. De facto esta modelação não é trivial e prende-se fortemente com o contexto a que se pretende aplicar a troca de informação e com os requisitos específicos dos parceiros. Até recentemente estes temas não tinham ainda sido abordados. Adicionalmente, a responsabilidade de implementação de mecanismos de interoperabilidade faz parte do Processo Produto ou Serviço (PPS) 2, e não do PPS1 em que me integro, e é responsabilidade de múltiplas organizações.

3.4 Formato de Comunicação entre Web Services

Os formatos típicos de troca de dados com *Web Services* são XML [33] e JSON [34], sendo que existem numerosos fornecedores de APIs que suportam simultaneamente ambos os

formatos. O XML é uma linguagem de *markup*[5] que define um conjunto de regras para codificação de documentos num formato legível por máquinas e humanos. O formato JSON é um *standard* aberto para troca de dados baseado em texto, que se destaca por ser legível e leve. O XML é, de longe, o formato de dados mais popular para APIs *Web*, embora o JSON tenha vindo a crescer continuamente em popularidade e uso. Muitos programadores preferem o formato JSON, entre outras razões, porque: os dados podem ser lidos sem um *parser* complexo na maioria das linguagens de programação modernas; existem *parsers* para JSON em quase todas as linguagens de programação; os *parsers* de JSON são rápidos, pequenos e simples; os dados em JSON são muito menores que em XML porque o JSON não inclui *namespaces*, atributos, entre outros elementos (como faz o XML) [35].

Os dados de configuração que se pretende guardar ficariam mais legíveis e seriam mais facilmente trabalhados por um ser humano usando XML, contudo uma vez que hoje em dia já existem *parsers* automáticos para converter objetos em XML e JSON, e vice-versa, o JSON será usado para serialização e transmissão dos dados estruturados uma vez que é mais leve que o XML. Adicionalmente, é muito mais fácil e eficiente trabalhar com JSON do que com XML utilizando JavaScript, e tanto o visualizador de dados como o painel de administração e editor de entidades têm uma forte vertente do lado do cliente utilizando JavaScript. Esta escolha é ainda suportada pelo facto de que atualmente o JSON é de tal forma popular entre programadores *web* que alguns fornecedores de APIs disponibilizam exclusivamente este formato (como por exemplo: o Foursquare, o Twitter e o Google+) [35].

3.5 Análise de Sistemas de Gestão de Bases de Dados Relacionais *Versus* Não Relacionais no Contexto deste Projeto

A investigação a esta questão foi extensa e detalhada e pode ser consultada no documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)", na secção A.3.5 Análise de Alternativas de Armazenamento Persistente.

Após a análise de soluções típicas mantém-se a questão do tipo de base de dados a utilizar que é respondida e justificada nesta secção. A conclusão deste capítulo apresenta a avaliação da escolha feita tendo em conta as últimas reuniões do TICE.Healthy.

Viver na vanguarda tecnológica é uma perspetiva motivadora, mas que deve ser abordada com extrema precaução. Apesar de promissoras, as bases de dados NoSQL ainda enfrentam diversos desafios. A sobrecarga e complexidade que tipicamente advém de não se poder executar uma *query* SQL, a não existência de restrições Atomicidade, Consistência, Isolamento e Durabilidade (ACID), a falta de familiaridade com a tecnologia e a ecoestrutura limitada são, entre outros, motivos para dúvida e para preocupação.

Numa fase inicial, dada a análise do problema já realizada, a base de dados NoSQL que se parece adequar melhor aos requisitos do TICE.GenericEntity e responder melhor às dúvidas e preocupações à volta do movimento NoSQL, seria o MongoDB. As principais razões para esta seleção são listadas no apêndice A3.2 Razões para Seleção do MongoDB dentro das Escolhas NoSQL do documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)".

Apesar das motivações constatadas poderá ser precipitado usar uma base de dados NoSQL sem ter um problema específico e existem autoridades da área que defendem que a

[5] Linguagem de *markup* é um sistema para anotar texto de uma forma que é sintaticamente distinguível desse texto.

procura de alternativas a bases de dados relacionais deve acontecer quando existe um problema real a resolver [36]. De facto, as bases de dados NoSQL estão a tornar-se cada vez mais populares e, quando usadas de forma apropriada, podem oferecer benefícios reais. Contudo, devem não só ter-se em conta os desafios que estas ainda enfrentam [37], como o facto de que as bases de dados relacionais (tirando o MySQL) escalam muito mais verticalmente do que a maioria da comunidade científica gosta de admitir [38]. Embora seja tentador experimentar novas ferramentas para resolver um problema, existe uma curva de aprendizagem envolvida tanto em desenvolvimento como operações [36]. É verdade que neste cenário os requisitos de extensibilidade e flexibilidade nos orientam para uma solução sem esquema, mas não é verdade que a preocupação atual no módulo a ser desenvolvido seja escalabilidade de ponta *on-demand* que não pode ser conseguida simplesmente escalando verticalmente. Esta última vantagem, que por enquanto se revela desnecessária neste cenário, é reconhecida como a principal força motivadora do movimento NoSQL.

O risco que advém do desvio da escolha (relacional) segura é bem real. Tomando como exemplo a solução NoSQL considerada favorita para este cenário, o MongoDB, podem-se encontrar tanto histórias de sucesso (como a empresa conimbricense Bundlr [39], que arriscou no MongoDB [40], e a Boxed Ice que, mesmo com algumas dificuldades [41], ainda hoje usa o MongoDB em produção e se encontra satisfeita [42]), como casos de advertência para o perigo destas soluções (como é exemplo a empresa Urban Airship [43] que voltou ao PostgreSQL após uma experiência desastrosa com o MongoDB [44], [45] e do Foursquare [46] que teve um incidente que o deixou inativo por 11 horas [47], [48]). A mensagem destes e outros exemplos acaba por ser bastante clara: é um risco apostar em NoSQL nesta fase, mas tipicamente essa decisão é tomada para lidar com problemas reais, como foi o caso da Urban Airship, porém a equipa de desenvolvimento precisa de entender as decisões de *design* do sistema que estão a escolher e os compromissos associados [49], [50].

Adicionalmente existem questões de negócio a considerar. A que mais pesa nesta escolha é a existência de grande preocupação por parte da gestão do TICE em manter as tecnologias e sistemas utilizados no projeto tão uniformes quanto possível para que seja mais fácil vender a plataforma. Com efeito, esta torna-se menos apelativa e menos vendável se for necessário juntar uma equipa altamente multidisciplinar para tratar da sua manutenção. Uma vez que existem componentes do sistema que vão utilizar PortgreSQL como base de dados considera-se uma grande vantagem para o negócio que tantos componentes quanto possível utilizem a mesma solução. A HIS [51], empresa líder do TICE.Healthy, também revelou preferência na utilização de uma base de dados relacional. Como tal, foi decidido que, por agora, este módulo irá utilizar a base de dados relacional PostgreSQL, mas que, dentro do possível, se continuarão a investigar alternativas tanto para se analisar se esta foi ou não a melhor decisão, como para concluir se será justificável migrar o sistema no futuro.

3.6 Motores de Regras de Negócio

De forma a permitir que a Generic Entity seja extensível e flexível e verdadeiramente genérica em ambiente de produção é necessário providenciar mecanismos para adicionar e alterar regras de negócio do núcleo da aplicação. A principal motivação desta necessidade está em poder validar um ou mais campos de uma entidade de formas que não foram previstas inicialmente. Por exemplo, se tivéssemos dois campos de uma entidade que fossem datas e uma das regras de validação fosse que a segunda data tinha que ser posterior à primeira e tal não tivesse sido previsto ou implementado do lado do servidor, deveria ser possível adicionar uma regra para validar essa condição. Para além deste caso básico existem muitos outros que podem contar com o motor de regras para tornar a aplicação mais adaptativa a mudanças sem quaisquer custos de recompilação ou *redeploy*.

As validações mais comuns e previsíveis vão ser previamente programadas do lado do servidor por uma questão de maior conveniência e facilidade de configuração e devido a custos de desempenho que inevitavelmente incorrem do uso de regras dinâmicas. Isto é, não se justifica que todas as regras de negócio sejam configuradas utilizando o motor de regras, só as de natureza dinâmica ou imprevistas.

Um motor de regras ajuda a reduzir os problemas e dificuldades inerentes ao desenvolvimento e manutenção da lógica de negócio de uma aplicação, por exemplo os custos de recompilação e *redeploy*. A maioria dos motores de regras permite usar programação declarativa para expressar consequências que são válidas dada determinada informação ou conhecimento. Assim, é possível concentrar nos factos que se sabem ser verdadeiros e nos resultados associados, isto é, na lógica de negócio da aplicação [52]. Estes motores podem ser utilizados para resolver problemas extremamente complexos, mas adequam-se igualmente bem a sistemas de validação de formulários [53] que é o seu objetivo principal neste projeto. Com efeito, casos de uso típicos de motores de regras incluem validação de dados e invocação de serviços [54].

Uma situação típica que sugere o uso de um motor de regras de negócios é o código de lógica de negócio conter uma série de instruções *if-else*. Um motor de regras não só pode ajudar a organizar esta lógica expressando-a de uma forma declarativa como também pode substituir condições *if then* por uma rede otimizada.

Afigura-se que neste contexto a principal desvantagem desta solução é que poderá existir redundância de armazenamento de regras (por exemplo se a mesma regra se aplicar a entidades diferentes). O desempenho não deverá ser problemático, uma vez que se pode guardar as regras compiladas bastando carregar as mesmas se estas não tiverem sido alteradas. Ainda assim, o uso do motor de regras na construção ou manutenção de uma aplicação que use como base a Generic Entity é totalmente opcional, dado que para um projeto com necessidades específicas estas podem ser previamente programadas.

3.6.1 Critérios para Escolha do Motor de Regras

Os critérios utilizados para escolha do motor de regras foram os seguintes:

- Estarem em desenvolvimento ativo, isto é, motores cujo desenvolvimento e suporte não tenham sido descontinuados.
- Ser um motor Java para mais fácil integração e aprendizagem.
- Ser uma tecnologia comprovada em produção e considerada escalável e com bom desempenho.
- Ter licença de *software* livre.
- Ser principalmente orientado para gestão de regras de negócios e não outro âmbito.

3.6.2 Escolha do Motor de Regras

Existem dezenas de motores de regras bem conhecidos, contudo a escolha foi relativamente simples aplicando os critérios apresentados que excluíram a maioria das opções de uma análise mais profunda. Para começar, aplicando o primeiro critério, a escolha é significativamente reduzida já que muitos projetos foram abandonados nos últimos anos (por exemplo as ferramentas Java: Jamocha, cuja última versão beta foi disponibilizada há 3 anos; o Zilonis, que não apresenta atividade desde 2007; o JRuleEngine, cuja última versão é de 2008; o jDREW, que não apresenta atualizações ao código desde 2006; entre muitos outros). Ao requerer que o motor tenha licença livre são descartados motores extremamente

promissores e líderes de mercado como o ILOG JRules [55–57] e Blaze Advisor [58]. O WF Rules [59] é uma das melhores escolhas gratuítas, mas tem como ambiente de desenvolvimento o .NET, logo também foi descartado.

Apesar de ter uma licença aberta e poder ser utilizado em ambiente de desenvolvimento Java, o Esper [60] é por natureza um motor de *Complex Event Processing* e não de construção de sistemas baseados em regras.

O Jena [61] é *open-source*, tem como ambiente de desenvolvimento o Java e contém um motor de regras especialmente poderoso em questões de *Web* Semântica. Contudo, o seu âmbito não é focado em contextos de regras de negócio, mas sim na programação para *Web* Semântica, pelo que também foi excluído [62].

O OpenRules [63] também é fácil de integrar com Java, tem licença aberta e boa documentação (para além de ser simples de perceber mesmo por não programadores), mas é uma ferramenta comparativamente fraca: apenas suporta definição de regras através de tabelas de decisão e tem uma comunidade pouco ativa.

Dadas estas considerações, o motor de regras escolhido (e o único a satisfazer todos os critérios) foi o Drools (JBoss Rules) [64]. Este é:

- Suportado por uma comunidade ativa.
- Fácil de utilizar e versátil, podendo ser embebido código Java diretamente nas regras.
- Rápido a executar (utiliza o algoritmo Rete-OO, para melhor integração com Java, baseado no Rete[6], que é superior a abordagens ingénuas que verificam todas as condições).
- Está em conformidade com a API Java Rule Engine (JSR-94), o que melhora a compatibilidade e facilidade de integração com outros sistemas.

Para além disso salienta-se que o Drools é o projeto *open-source* mais popular do mercado estando aplicado em diversos ambientes de produção [57], [65].

3.7 Conclusão

Atualmente não existe uma solução de PHR em Portugal como a que se pretende implementar. Os serviços *web* mais populares (como o Google Health e o Microsoft Health Vault) não estão disponíveis neste país e o Sapo Saúde não oferece algumas funcionalidades chave como: vistas diferentes para utentes e profissionais de saúde, vistas distintas consoante o contexto específico dos utentes (faixa etária, se sofrem de alguma patologia, etc.) e um considerável ecossistema de aplicações para interagir e enriquecer o PHR.

No que toca a extensibilidade da Generic Entity, foram estudadas as formas como outros sistemas, de natureza distinta, alcançaram este objetivo, sendo possível distinguir fatores comuns como a necessidade de definir e guardar configurações e metadados para as entidades genéricas criadas.

Foi ainda considerado que, apesar de todas as soluções encontradas utilizarem bases de dados relacionais, o módulo pretendido poderia beneficiar de uma abordagem não relacional que conseguiria oferecer vantagens como: maior potencial de escalabilidade a um custo mais baixo e facilidade de alteração do modelo de dados (já que não seriam necessárias operações como por exemplo *ALTER TABLE*). Contudo, apesar de terem sido

[6] O Rete é um algoritmo de *pattern matching* eficiente para a implementação de sistemas de regras em produção [85].

reconhecidas vantagens, surgiram também preocupações a considerar na escolha da tecnologia a utilizar.

Tendo em conta as últimas reuniões de parceiros tornou-se evidente que a escolha relacional foi a mais correta no contexto da prova de conceito PHR. De facto, se for utilizado algum modelo de referência, como o RIM, este será construído através das contribuições dos vários parceiros e irá certamente usar um Sistema de Gestão de Bases de Dados Relacional. Se for realmente este o caso, a Generic Entity já está pronta para ser integrada utilizando as tabelas personalizadas que forem criadas. Para além disso algumas responsabilidades específicas do PHR, como a segurança e privacidade, terão contribuição da HIS, e esta, tal como já foi referido, mostrou preferência por bases de dados relacionais e poderia ter mais dificuldade em estudar os conceitos e utilizar a ferramenta se a decisão tivesse sido diferente. Contudo, a camada de abstração criada para a fonte de dados (que será apresentada no Capítulo 5) não deixa de ser útil para separar responsabilidades e para permitir a implementação da uma nova camada, caso se pretenda utilizar outro Sistema de Gestão de Bases de Dados (SGBD) num projeto distinto.

Importa ainda responder mais concretamente à questão levantada na defesa intermédia sobre o uso de uma Triple Store. Apesar de apresentar capacidades de extensibilidade e flexibilidade muito superiores a uma base de dados relacional, não seria possível juntar a solução Generic Entity a um SGBD relacional comum utilizando esta abordagem, isto é, deixaria de ser viável usar uma tabela de base de dados normal criada por um parceiro e configurá-la com a Generic Entity. No primeiro semestre não era sabido que a base de dados poderia provir da contribuição de vários parceiros e por isso provavelmente seria relacional (logo o MongoDB ainda era uma alternativa viável) contudo, já tinha havido a informação específica que não era pretendida a utilização de uma Triple Store. Adicionalmente, a principal razão para excluir a Triple Store, é que esta não parece encaixar-se no caso de uso típico do PHR. De facto, uma Triple Store é uma base de dados especificamente otimizada e construída para armazenar e recolher triplos, quando o que se pretende principalmente com esta solução é armazenar e recuperar registos inteiros de uma só vez. Por fim, sendo este um projeto que poderá servir de base a múltiplos outros sistemas faz sentido que satisfaça, logo à partida, as necessidades e escolhas mais comuns (que claramente, no contexto das bases de dados, é a abordagem relacional) e que deixe as implementações mais específicas (como as que necessitam de Triple Store) para serem criadas pelos projetos que delas necessitem. O ideal seria que implementações específicas de novos projetos estivessem disponíveis para serem reutilizadas quando voltassem a ser necessárias e tem havido esforços nesse sentido dentro do IPN.

Finalmente, foi reconhecido que os benefícios de um motor de regras tornam a sua integração uma mais-valia para o projeto. Concretamente, as principais vantagens que motivam a sua necessidade são a possibilidade de adicionar e alterar as regras de negócio sem recompilação de toda a aplicação, a facilidade de comunicar as regras que se tornam mais legíveis e fáceis de trabalhar que código Java (especialmente se forem programadas de forma a serem tão legíveis quanto possível, ou, por ventura, se se integrar uma ferramenta para definir as regras visualmente) e a possibilidade de ter uma melhoria de desempenho para conjuntos grandes de regras (o que poderia ser interessante para implementar mecanismos como por exemplo alertas de saúde). Com efeito, o problema base de validação de entidades não é complexo, mas não há uma forma adequada a todos os casos que preveja todas as necessidades futuras do sistema e o permita ser programado de forma fixa e intemporal num sistema dinâmico e configurável.

A investigação realizada serve de base à definição dos requisitos e às escolhas e desenho da arquitetura a apresentar nas secções seguintes.

Capítulo 4
Requisitos

O principal objetivo deste estágio foi desenvolver um sistema que permitisse mapear um esquema de base de dados numa interface para o utilizador. Este sistema reflete automaticamente, em todas as suas camadas, modificações feitas às definições das entidades genéricas ou dos menus criados que definem os componentes a apresentar na página *web* do projeto em que esteja integrado.

O propósito deste capítulo é a identificação e descrição dos requisitos do módulo a ser desenvolvido, para que possam ser conhecidos, compreendidos e aprovados por outros parceiros do projeto.

Para a definição destes requisitos foram avaliadas as necessidades das partes interessadas (*stakeholders*) e foram estudados projetos semelhantes como: o Meu Sapo Saúde, Google Health e diversos sistemas de criação e alteração dinâmica de formulários como o Microsoft SharePoint.

A versão completa deste documento está disponível no ficheiro anexo intitulado "TICE-GenericEntity Análise de Requisitos".

4.1 Análise de Atores

Os principais atores que foram considerados na análise de casos de uso do módulo TICE.GenericEntity foram:

Fornecedores de Serviços

Este ator representa todas as entidades que providenciam serviços ao utilizador final (por exemplo o portal e as aplicações por este providenciadas).

Administrador

Este ator representa um utilizador com privilégios de gestão máximos sobre a plataforma.

Utilizador

Este ator representa um utilizador dos serviços disponibilizados como por exemplo, no caso do PHR, um utente ou profissional da área de saúde.

4.2 Levantamento de Requisitos

O levantamento de requisitos foi particularmente difícil uma vez que não existem clientes (pelo menos não no sentido clássico) para a plataforma do TICE.Healthy. Contudo existem *stakeholders*, todos eles parceiros na iniciativa. Adicionalmente, alguns requisitos estão dependentes de outros módulos a serem desenvolvidos no projeto pelo que foi impossível prever se seriam realizados, e como seriam realizados, dentro do período de estágio. Para este módulo específico os requisitos vieram das seguintes fontes:

- Gestores da iniciativa TICE, destacando-se o *Product Owner*, Engenheiro Alcides Marques, e o chefe da equipa do IPN do TICE.Healthy, Mestre João Quintas.

- Análise do estado da arte.
- Plano de negócios do TICE.Healthy.
- Empresa HIS, líder do projeto TICE.Healthy.

O levantamento realizou-se não só por análise de outras soluções e de documentação pré-existente do projeto, mas também através de:

- **Reuniões** que reduziram as omissões, ambiguidade da informação e volume de documentação formal para comunicação entre as partes e permitiram evitar entrevistas pouco eficientes.
- **Protótipo** desenvolvido para apresentar numa reunião de gestão de forma a alinhar visões sobre o projeto e serem identificadas quaisquer falhas de interpretação o mais cedo possível.

4.3 Análise de Requisitos

Os requisitos foram divididos em requisitos da Generic Entity e requisitos do editor de entidades da Generic Entity. Em cada um são apresentados os requisitos funcionais e não funcionais. Neste documento vão ser listados apenas os requisitos da Generic Entity, uma vez que a lista completa de requisitos é extensa.

Os requisitos contêm os seguintes campos:

- **Código** que identifica unicamente o requisito.
- **Versão** do requisito.
- **Tipo** de requisito. Este campo só se aplica aos requisitos não funcionais (e destes apenas a alguns) e identifica a sua categoria. Por exemplo, um requisito não funcional pode ser do tipo Usabilidade, Segurança, Desempenho, entre outros.
- **Título** do requisito.
- **Descrição** do requisito (quando não for evidente apenas pelo título).
- **Atores** diretamente ligados a este requisito (este campo só é incluído quando considerado adequado).
- **Prioridade** atribuída ao requisito que segue o método de MoSCoW[7].

Os requisitos cujo título se encontra riscado não foram realizados. A listagem feita não contempla os requisitos do editor de entidades desenvolvido. Estes podem ser consultados no documento "TICE-GenericEntity Análise de Requisitos".

4.3.1 Requisitos Funcionais

Tabela 4 Listagem de requisitos funcionais da Generic Entity.

Código	Versão	Título	Atores	Prioridade
GEF01	1.0	Definição e Armazenamento de Entidades/Formulários	Administrador	*Must*
Possibilidade de criar novas entidades e armazenar a sua definição na base de dados.				

[7] **M**o**SC**o**W** [86] é uma técnica de priorização de requisitos pela utilização de palavras com significado. O esquema definido é: ***M****ust* – tem que ter este requisito para satisfazer os requisitos de negócio – ***S****hould* – deve ter este requisito se possível, mas o sucesso do projeto não depende dele – ***C****ould* – pode ter este requisito se não afetar o resto do projeto – ***W****on't* – não vai ter este requisito, mas poderá ser implementado no futuro.

Código	Versão	Título	Atores	Prioridade
GEF02	1.0	Versionamento de Entidades	Administrador	*Must*
O sistema permite a redefinição de entidades (extensão com mais campos, apagamento de campos e edição de propriedades e tipos de campos), sem perda dos dados antigos.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF03	1.0	Geração Automática de Formulários	Administrador	*Must*
Geração automática de formulários personalizados para visualização, inserção e edição de dados.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF04	1.0	Campos para Anexo de Ficheiros	Administrador	*Must*
Suporte de campos de *upload* de ficheiros e lidar com o seu armazenamento de forma transparente e automática.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF05	1.1	Armazenamento de Versões de Anexos	Utilizador	*Could*
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF06	1.1	Histórico de Ações e Acessos	Utilizador	*Could*
Permite visualizar o histórico de ações (*log* de quem visualizou e/ou alterou registos e quando).				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF07	1.0	Definição de Vistas Sobre os Dados	Administrador	*Must*
Permite definir e utilizar diferentes vistas sobre os dados (exemplo listar registos, novo, editar, visualizar) determinando que campos aparecem e como.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF08	1.1	~~Formulários de Várias Páginas~~	Administrador	*Won't*
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF09	1.0	~~Permissões nas Vistas Sobre os Dados~~	Administrador	*Could*
Possibilidade de definição de vistas controladas por permissões.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF10	1.0	Multilíngue	Administrador	*Could*
É possível definir mais que uma língua na configuração de entidades e o sistema gera os formulários de acordo com a língua escolhida.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF11.1 a 11.10	1.0	Campos que Suportam Inserção dos Tipos de Dados Mais Comuns	Administrador	*Must*
Campos de inserção de texto, *strings*, inteiros, *floats*, *doubles*, números, *longs*, moeda, data e hora.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF12	1.0	*Lookup Fields*[8]	Administrador	*Must*
Suportar a inserção de *lookup fields*/campos de pesquisa, que essencialmente permitirão definir relações 1:N entre entidades diferentes.				

[8] Campo de uma entidade para pesquisar/listar informação de outra entidade. Como exemplo imaginando que tínhamos uma entidade para representar doenças e uma para antecedentes, um *lookup field* na entidade antecedentes serviria para pesquisar e associar uma doença no formulário de adição de um novo antecedente.

Código	Versão	Título	Atores	Prioridade
GEF13	1.0	*Lookup Fields* com Possibilidade de Escolha de Vários Campos	Administrador do portal	*Could*
Suportar a inserção de *lookup fields*/campos de pesquisa com possibilidade de escolha de vários campos, que essencialmente permitirão definir relações N:N entre entidades diferentes.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF14	1.0	Instruções/Ajuda nos Formulários	Administrador	*Must*
Suportar inserção de instruções/ajuda nos formulários.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF15	1.0	Personalização de Propriedades de Campos *Web*	Administrador	*Must*
Poder definir propriedades como por exemplo qual o tipo de dados, tamanho máximo, padrões de validação, entre outras.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF16	1.0	Suporte Para Execução de *Queries Ad-hoc*	Administrador	*Must*
Suporte para execução de *queries ad-hoc* sobre os dados armazenados.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF17	1.0	~~Suporte para funcionalidades de *reporting*.~~	Administrador	*Could*
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF18	1.0	~~Suporte para Sistema de Pagamento~~	Administrador	*Won't*
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF19	1.0	Adição de Código Personalizado nos Formulários	Administrador	*Could*
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF20	1.0	~~*Branching* Lógico~~	Administrador	*Could*
Suporte para adição de lógica de *branching* (servir conteúdo diferente ao utilizador com base nas suas respostas).				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF21	1.0	~~Envio de *Emails*~~	Administrador	*Could*
Possibilidade de definição de envio de *emails* personalizados aquando a submissão de um formulário.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF22	1.0	Definição de Ações Despoletadas por Eventos	Administrador do portal	*Should*
Possibilidade de definição de ações personalizadas despoletadas por eventos (para além das embutidas, como guardar na base de dados).				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF23	1.1	Exportação de Dados em Múltiplos Formatos	Utilizador	*Could*
Possibilidade de exportação de dados em múltiplos formatos: *Portable Document Format* (PDF), *Comma-Separated Values* (CSV), Excel.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF24	1.0	~~Vista de Inserção de Anotações~~	Utilizador	*Won't*

Código	Versão	Título	Atores	Prioridade
GEF25	1.1	Estender Sistema com Controlos *Web* Personalizados	Administrador	*Could*
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF26	1.1	~~Pesquisa de Conteúdo por Motor de Procura~~	Administrador	*Won't*
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF27	1.0	Configurar Pesquisa nas Vistas de Listagem	Administrador	*Must*
Possibilidade de configurar a ativação da procura de conteúdo nas vistas de listagem.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF28	1.0	Configurar Ordenação nas Vistas de Listagem	Administrador	*Must*
Possibilidade de configurar a ativação da ordenação do conteúdo por campos nas vistas de listagem.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF29	1.0	Configurar Ordenação nas Vistas de Listagem Por Mais do que Uma Coluna	Administrador	*Could*
Possibilidade de configurar a ativação da ordenação do conteúdo por mais do que um campo/coluna nas vistas de listagem.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF30	1.0	Gerir Propostas	Administrador	*Must*
O sistema deve ter uma API para receber propostas de criação ou edição de entidades. Estas propostas devem poder ser geridas (pré-visualizadas, aceites ou rejeitadas com notificação por email) através do painel de administração.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF31	1.0	Configuração de Menus da Aplicação de Visualização	Administrador	*Must*
Deve ser possível configurar os menus e submenus da aplicação de visualização através do painel de administração.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF32	1.0	Adicionar Componentes a um Menu	Administrador	*Must*
Deve ser possível adicionar componentes a um menu (esses componentes podem ser: vistas da Generic Entity, conteúdo inserido pelo administrador, conteúdo inserido por *iframe* ou ainda conteúdo servido por uma aplicação através do URL).				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF33	1.0	Relacionar Componentes	Administrador	*Must*
Deve ser possível configurar relações entre componentes para que a manipulação de um componente leve a que outros que estão à escuta tomem ações.				
Código	**Versão**	**Título**	**Atores**	**Prioridade**
GEF34	1.0	Inserir e Editar Conteúdo Definido pelo Administrador	Administrador	*Must*
Deve ser possível ao administrador definir e editar novo conteúdo para a aplicação de visualização. Esse processo terá pré-visualização embutida e os efeitos da manipulação do conteúdo devem ser imediatos em ambiente de produção.				

4.3.2 Requisitos Não Funcionais

Tabela 5 Listagem de requisitos de produto.

Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS1	1.1	Segurança	~~Ligações Seguras~~	*Could*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS2	1.0	Segurança	Validação de *Input* no Servidor	*Must*
Serão realizadas verificações e validações do lado do servidor (*input* externo será sempre validado).				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS3	1.0	Segurança	Impedir SQL *Injection*	*Must*
Os SQL *statements* típicos das linguagens de programação são vulneráveis a SQL *injection*. Utilizar *prepared statements* e adicionar valores como parâmetros e não como parte da *string*/comando a executar.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS4	1.0	Segurança	Sanitizar *Input* do Utilizador	*Must*
Sanitizar input do utilizador para evitar ataques como *Cross-Site Scripting*.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS5	1.0	Segurança	Impedir Acessos Não Autenticados	*Must*
A aplicação de visualização não pode ser acedida antes do utilizador estar devidamente autenticado.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS6	1.0	Segurança	Impedir o Acesso de Utilizadores Comuns ao Painel de Administração	*Must*
A zona de administração só pode ser acedida por utilizadores administradores.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS7	1.0	Segurança	Impedir Ataques *Cross-Site Request Forgery* (CSRF)	*Must*
Gerar e verificar *token* de autenticidade para evitar ataques CSRF.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS8	1.0	Segurança	~~Encriptar Dados na Base de Dados~~	*Won't*
Cifrar dados na base de dados com uma chave pública para a qual apenas a aplicação Core dispõe da respetiva chave privada (protegendo assim também de ataques externos ao excluir os administradores de sistemas).				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS9	1.0	Segurança	~~Garantir Não Repúdio~~	*Won't*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS10	1.0	Segurança	Não Guardar os *Tokens* OAuth dos Utilizadores	*Should*
Pedir ao portal o *token* de acesso quando é necessário apenas para obter dados iniciais e descartar logo que possível.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS11	1.0	Segurança	Sessões Assinadas	*Must*
Manter as sessões por *cookies* assinados para garantir que o seu conteúdo não é forjado.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPS12	1.0	Segurança	Chave Secreta Usada para Assinar Sessões	*Should*
O segredo utilizado para assinar a sessão é privado e tem que ser criado um novo quando é colocado em produção (esse segredo não pode ser armazenado no Git).				

Código	Versão	Tipo	Título	Prioridade
GENFPL 1	1.0	*Logging*	Registo de Exceções	*Must*
Todos os erros ou exceções que ocorram devem ser registados num ficheiro específico.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPL 2	1.0	*Logging*	Registo de Acessos	*Could*
Todos os acessos ao módulo devem ser registados.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD 1	1.0	Documentos	Estado da Arte	*Must*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD 2	1.0	Documentos	Análise dos Requisitos	*Must*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD 3	1.0	Documentos	Arquitetura do Sistema	*Must*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD 4	1.1	Documentos	Manual do Programador da Generic Entity	*Must*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD 5	1.0	Documentos	Análise de Segurança	*Must*
Documento em que se analisam questões de segurança na Generic Entity.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD E1	1.0	Escalabilidade	Suporte para Balanceamento de Carga	*Must*
O módulo irá permitir a utilização do padrão de balanceamento de carga (isto implica que as aplicações sejam *stateless*).				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD E2	1.0	Desempenho e Escalabilidade	Utilização de Tecnologias Comprovadas	*Must*
O módulo irá utilizar unicamente tecnologias (bases de dados, *frameworks*, servidores *Web*, etc.) que apresentem exemplos de utilização com sucesso em ambientes de produção.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD E3	1.1	Desempenho	Desempenho na Geração e Apresentação de Formulários	*Should*
A geração e apresentação automática de formulários não pode demorar mais que duas vezes o tempo de geração e apresentação de formulários não dinâmicos com a tecnologia escolhida (utilizando quaisquer mecanismos de melhoria de desempenho considerados adequados).				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPD E4	1.0	Desempenho e Escalabilidade	Validações do Lado do Cliente	*Must*
As validações são realizadas também do lado do cliente para evitar ao máximo o peso do servidor lidar com submissões inválidas.				

Código	Versão	Tipo	Título	Prioridade
GENFPDE5	1.0	Desempenho e Escalabilidade	Suporte para Programação Assíncrona na Aplicação Viewer para Comunicação com o Core	*Should*
Em *frameworks* de desenvolvimento *Web* que sejam *event-driven* existe um número de *threads* limitado para satisfazer pedidos e um pressuposto que os pedidos são curtos. Quando este não é necessariamente o caso deve-se usar programação assíncrona para que as *threads* não fiquem bloqueadas prejudicando o desempenho e comprometendo a escalabilidade.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPDE6	1.0	Desempenho e Escalabilidade	Renderização de Templates do Lado do Cliente	*Should*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPDE7	1.0	Desempenho e Escalabilidade	Combinação, Minificação e Compressão de JavaScript	*Should*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPDE8	1.0	Desempenho e Escalabilidade	Combinação, Minificação e Compressão de CSS	*Should*
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPU1	1.1	Usabilidade	Validações do Lado do Cliente *On Blur*[9]	*Must*
As mensagens de erros no preenchimento de formulário serão apresentadas durante ou após o preenchimento de cada campo (e não apenas quando se clica na submissão).				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPU2	1.0	Usabilidade	Focar Primeiro Elemento Inválido no Caso de Submissão Inválida	*Should*
Após tentar submeter um formulário inválido, o primeiro elemento inválido é focado, permitindo ao utilizador corrigi-lo. Se outro campo inválido, que não o primeiro, estava focado antes da submissão, então é esse o campo focado, permitindo ao utilizador começar do fundo se é assim que prefere.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPU3	1.0	Usabilidade	Validação Não Ativa Antes de Haver Hipótese de ser Inserido o Valor Correto	*Must*
Antes do campo ser marcado como inválido a validação é "preguiçosa". Antes de submeter o formulário pela primeira vez, o utilizador pode percorrer os campos sem receber mensagens inconvenientes. Isto é, o utilizador não é incomodado antes de ter a hipótese de inserir o valor correto.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPU4	1.0	Usabilidade	Validação Ávida	*Must*
Após um campo ser marcado como inválido é avidamente validado. Assim que o utilizador insira o valor necessário a mensagem de erro é removida.				

[9] *On blur* é o evento de deixar de ter o foco numa componente (neste caso será num campo do formulário).

Código	Versão	Tipo	Título	Prioridade
GENFPU 5	1.0	Usabilidade	Mensagens de Falha Adequadas	*Must*
Falhas de acesso ao sistema (por exemplo registo inexistente ou falha de acesso a base de dados) causam uma mensagem de aviso apropriada para o utilizador explicando que a operação não pôde ser concluída.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPU 6	1.0	Usabilidade	Submissões por AJAX	*Must*
Os dados submetidos através de um formulário (mesmo que contenha anexos) são enviados para o servidor por AJAX para que não ocorra um *refresh* da página. É recebida uma mensagem a indicar o sucesso ou insucesso da operação no final da submissão.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPU 7	1.0	Usabilidade	Lidar com Latência de Submissão	*Must*
Ao submeter dados por formulário este apresentará um elemento gráfico durante o processo de envio que informe o utilizador de que o processo está em curso. O botão de submissão é desativado imediatamente após o início da submissão, durante o envio, para que o utilizador não tente fazer múltiplas submissões por engano.				
Código	**Versão**	**Tipo**	**Título**	**Prioridade**
GENFPU 8	1.0	Usabilidade	*Layout* Fluído e Responsivo	*Should*

Tabela 6 Listagem de requisitos organizacionais.

Código	Versão	Título	Prioridade
GENFO1	1.0	Uniformização das Tecnologias	*Must*
Dentro da plataforma em que se insere o módulo as tecnologias devem ser uniformizadas tanto quanto possível para a tornar mais vendável e para que os custos de manutenção futuros sejam reduzidos.			
Código	**Versão**	**Título**	**Prioridade**
GENFO2	1.0	*Backends* em Java	*Must*
A gestão do TICE decidiu que qualquer componente de *backend* da plataforma tem que ser criado utilizando a linguagem de programação Java. Esta regra inclui o *backend* do módulo Generic Entity.			
Código	**Versão**	**Título**	**Prioridade**
GENFO3	1.0	PostgreSQL como SGBDR da Plataforma	*Must*
Se for utilizada um sistema de gestão de dados relacional este terá que ser o PostgreSQL.			
Código	**Versão**	**Título**	**Prioridade**
GENFO4	1.0	Documentação em Língua Portuguesa	*Must*
Toda a documentação produzida nos TICE (salvo exceções definidas por órgãos da gestão) terá que ser escrita em Português.			
Código	**Versão**	**Título**	**Prioridade**
GENFO5	1.0	Código Fonte e Comentários em Inglês	*Must*

Tabela 7 Listagem de requisitos externos.

Código	Versão	Título	Prioridade
GENFE1	1.0	API para Fornecedores de Serviços	*Must*
Interoperabilidade com fornecedores de serviços (como por exemplo o portal) através de *application programming interface* (API).			
Código	**Versão**	**Título**	**Prioridade**
GENFE2	1.1	~~Integração do Módulo de Permissões~~	*Won't*
Código	**Versão**	**Título**	**Prioridade**
GENFE3	1.1	Integração com o Portal	*Must*
Integração com o portal (o PHR é uma aplicação do portal). Usar OAuth para obter dados do utilizador autenticado no portal e utilizar a API do portal para obter dados como a língua.			
Código	**Versão**	**Título**	**Prioridade**
GENFE4	1.0	Integração com o Sensor Care	*Should*
Integração com o Sensor Care para que este use a Generic Entity para persistir os dados e os retransmitir por XMPP.			

Conclusão

Os requisitos deste projeto são bastante dinâmicos e sofrem considerável influência de empresas parceiras da iniciativa TICE e também de questões de interoperabilidade com outros módulos a serem criados para a plataforma, razão pela qual sofreram algumas alterações.

Foram definidos vários requisitos que, apesar de interessantes e mesmo importantes para o futuro, têm menos relevância numa fase inicial do projeto (ou não podem ser implementados até que sejam tomadas determinadas decisões a nível da gestão do TICE.Healthy) e, como tal, tem interesse que estejam documentados, mas não se esperava que fossem todos implementados no contexto do estágio. Esta é a principal razão da escolha do método de MoSCoW para a priorização dos requisitos já que o seu esquema tem em conta requisitos *Could,* que só devem ser implementados se não afetarem o resto do projeto, e *Won't* que devem ficar documentados, mas que só serão respondidos fora do contexto do estágio.

Salienta-se que os requisitos *Must* (críticos para o sucesso do projeto) foram todos desenvolvidos.

Capítulo 5
Desenho e Análise

Este capítulo descreve a estrutura e conceitos associados à arquitetura geral do módulo Generic Entity.

O estado da arte permitiu dar forma ao módulo e determinar quais as abordagens mais adequadas para solucionar alguns problemas e, em conjunção com a análise de requisitos, suporta várias decisões tomadas.

5.1 Problema de Engenharia

O módulo a desenvolver neste estágio enfrentou desafios interessantes que são salientados nesta secção.

Para começar, o que se pretende é algo ambicioso, nomeadamente modelar em *runtime* a estrutura de dados que irá, por sua vez, moldar a interface. Estas capacidades darão resposta à grande necessidade de adaptação dinâmica e configuração em função dos requisitos. De facto, existe um grande peso associado a alterações de requisitos ou adição de funcionalidades que, tipicamente, obrigam não só à alteração da camada de acesso a dados e/ou de negócios, mas também, na maioria das vezes a alterações na interface. Estas alterações à camada de apresentação podem ser complicadas uma vez que cada tipo de campo necessita de um controlo visual diferente e a interface pode conter regras complexas de visibilidade e comportamento dos campos que dependem de aspetos como: permissões do utilizador, página anterior ao pedido, entre outras. O código a ser produzido manualmente neste tipo de problemas é repetitivo, o desenvolvimento é lento e potencialmente sujeito a muitos erros. Estas questões justificam a criação de uma ferramenta de geração de código que satisfaça as necessidades com o mínimo de programação e trabalho. Para além disso, em casos tradicionais, este tipo de alterações não pode ser aplicado diretamente em ambiente de produção, ou ser imediatamente refletido no *browser*, como seria desejável.

Dadas estas questões, a forma de resolver o problema não é rotineira. Para exemplificar este ponto podemos considerar que, em teoria, aplicando a abordagem escolhida de tabela genérica híbrida, é possível ter duas tabelas para armazenamento de milhares de entidades, em vez da situação mais tradicional que consiste em ter uma tabela por entidade.

Alguns obstáculos interessantes com que foi preciso lidar, são:

- A própria clarificação, definição e visão do problema a resolver.
- Permitir o nível de personalização necessário.
- Emular parte do comportamento de acesso à base de dados.
- Incluir um gerador de código (recebe configuração e gera o resultado desejado) com dois objetivos:
    - Acesso à base de dados.
    - Gerar interfaces (API e componentes *web*) de manipulação e visualização de

dados.

- Investigação de soluções aplicadas em produtos (fechados) e escolhas difíceis:
  - Que abordagens existem?
  - Como funciona este produto?
  - Qual a abordagem mais vantajosa?
  - Há alguma abordagem que responda por completo aos requisitos?
  - Qual a forma de armazenamento mais adequada?
- Desenhar a arquitetura e solução aplicando abordagens atípicas – perdendo as vantagens que advêm do uso das soluções convencionais – com a sensibilidade para escolher quando o fazer, e tendo sempre consciência das consequências que tais escolhas terão na implementação).
- A integração de uma série de novos elementos e enquadramento de eventos dinâmicos na construção e configuração de uma aplicação *web* de raiz.

Umas das conclusões do Estado da Arte é que existem várias abordagens implementadas no mercado que estão, essencialmente, erradas. Estas invertem completamente o que se pretende do sistema, seja por guardarem as configurações de entidades em campos de tabelas relacionais, comprometendo o desempenho e dificultando o acesso às definições de entidades, seja por guardarem os dados em XML ou JSON impedindo que se realizem *queries ad-hoc* simples e rápidas sobre esta informação. Neste estágio foi preciso ter a sensibilidade e espírito crítico para concluir que muitas abordagens comuns são desvantajosas, comprometem o desempenho, são difíceis de manter e são pouco flexíveis.

5.2 Descrição Geral da Plataforma

Numa perspetiva global, a plataforma We.Can desempenha um papel agregador no seio do projeto TICE.Healthy. Os seus objetivos principais prendem-se com a recolha e tratamento de dados de várias entidades colaboradoras para que possam ser partilhadas com os fornecedores de serviços [66].

Nesta secção aborda-se a arquitetura da plataforma a um nível conceptual, em conformidade com o documento de candidatura do projeto. Com base na especificação inicial da We.Can, é possível identificar duas funcionalidades/sistemas-chave que a plataforma deverá assegurar, nomeadamente o PHR e uma rede social [66], [67].

De uma forma geral, a plataforma deverá implementar funcionalidades nos seguintes domínios:

- Gestão de informação médica, social e de contexto.
- Integração de dispositivos móveis, controlo remoto e sensores.
- Segurança de informação.

A Figura 5 descreve, de forma esquemática, a composição da plataforma We.Can nos diferentes módulos que são propostos no projeto:

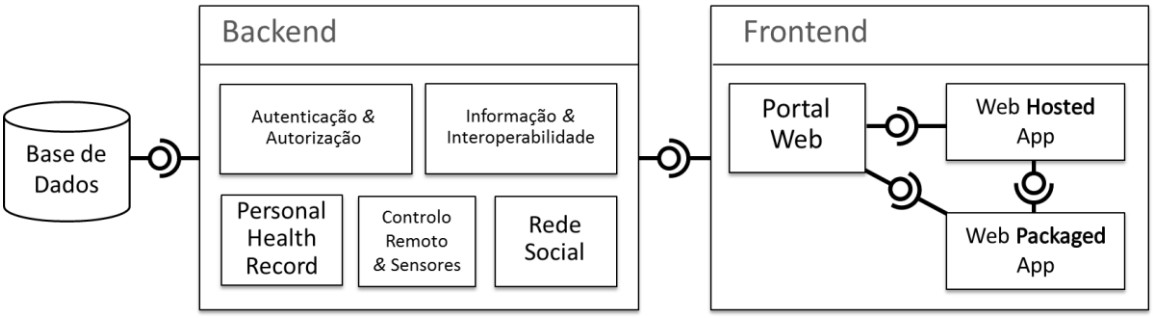

Figura 5 Arquitetura de alto nível do We.Can.

A Generic Entity é um módulo constituinte do PHR e irá interagir ainda com os outros módulos de formas que não estão ainda totalmente previstas, mas que irão ser apresentadas com o detalhe possível na secção 5.3.1 Visão de Nível 0.

Na Figura 5 são apresentados dois tipos de aplicações do portal. De forma sumária as aplicações *packaged* são executadas inteiramente no *browser* e a sua lógica de negócio é programada em JavaScript. Por outro lado, as aplicações *hosted* são acessíveis remotamente e alojadas fora da plataforma, sendo suportadas pelos seus próprios servidores. Para este último tipo de aplicação foi adotado o uso de *iframes* com capacidades adicionais. A necessidade de suportar dois tipos de aplicações deve-se ao facto de que embora a maioria das aplicações *packaged* seja mais responsiva e interativa, elas forçam o uso de um modelo de programação que pode nem sempre ser adequado ou desejado. De facto, programar a aplicação inteiramente em JavaScript pode ser restritivo em certos contextos.

O visualizador do PHR para consulta de dados clínicos e informais é uma aplicação *hosted*. Com efeito, esta aplicação cria vistas dinâmicas de acordo com as configurações armazenadas no repositório que têm em conta fatores como permissões do utilizador. Como resultado, a natureza desta aplicação impede-a de ser estritamente *client-side*. Contudo, é interessante notar que dados os últimos desenvolvimentos do portal é agora possível que a aplicação *web* PHR venha a ser *packaged*, em princípio sem grandes alterações. De facto, neste momento o portal já suporta que uma aplicação *packaged* aceda a um servidor externo, logo a aplicação *web* pode fazer pedidos para obter os seus componentes gerados do lado do servidor tendo em conta as configurações de permissões. Como a aplicação *web* configurada pela Generic Entity é *client-side* (programada em JavaScript) pode ser executada como aplicação *packaged*.

5.3 Descrição da Arquitetura

A arquitetura concebida para a plataforma é fortemente baseada nos princípios de desenho das arquiteturas orientadas a serviços[10], dada a sua natureza distribuída e extensível. Esta abordagem aumenta a abstração e encapsulamento dentro do sistema. A consequência mais interessante para o subprojecto TICE.GenericEntity, e outros subprojectos, é que gozam de maior autonomia no seu desenvolvimento, relativamente à plataforma completa.

De acordo com a análise de requisitos geral da plataforma é necessário implementar o protocolo de comunicação *Simple Object Access Proto*col (SOAP)[11] e o padrão de *software*

[10] Arquitetura Orientada a Serviços é uma abordagem distribuída que atribui cada função do sistema a um serviço independente e separado. Tal permite maior reutilização e flexibilidade quando comparado com sistemas altamente centralizados.
[11] SOAP é um protocolo para troca de informação baseado em XML sobre HTTP.

arquitetural *REpresentational State Transfer* (REST)[12], para comunicar com as fontes de dados. Contudo, apenas são usadas APIs REST para comunicar com os fornecedores de serviços.

A arquitetura do módulo é descrita com base em dois níveis de abstração. O nível 0 apresenta uma visão mais geral da ferramenta contemplando relações com agentes externos e o nível 1 apresenta com maior detalhe o seu funcionamento interno. Para cada um destes níveis são ainda apresentadas até três perspetivas, consoante se revelem necessárias, que são:

- **Perspetiva estática -** que aborda a organização do sistema em módulos ou componentes funcionais.
- **Perspetiva dinâmica –** que aborda o fluxo de dados dentro de módulos ou entre eles.
- **Perspetiva de *deployment* –** que documenta a apresentação física dos elementos do sistema (como servidores, ficheiros de código fonte, ficheiros de configuração, entre outros), em ambiente de *deployment* e/ou produção.

5.3.1 Visão de Nível 0

O objetivo deste nível é descrever o enquadramento da Generic Entity na interação com agentes externos ao seu funcionamento.

5.3.1.1 Perspetiva de deployment

Na perspetiva de *deployment* o sistema é contextualizado a nível físico, sendo identificados os agentes externos que com ele comunicam e as respetivas interfaces de comunicação. A Figura 6 apresenta a perspetiva de *deployment* de nível 0.

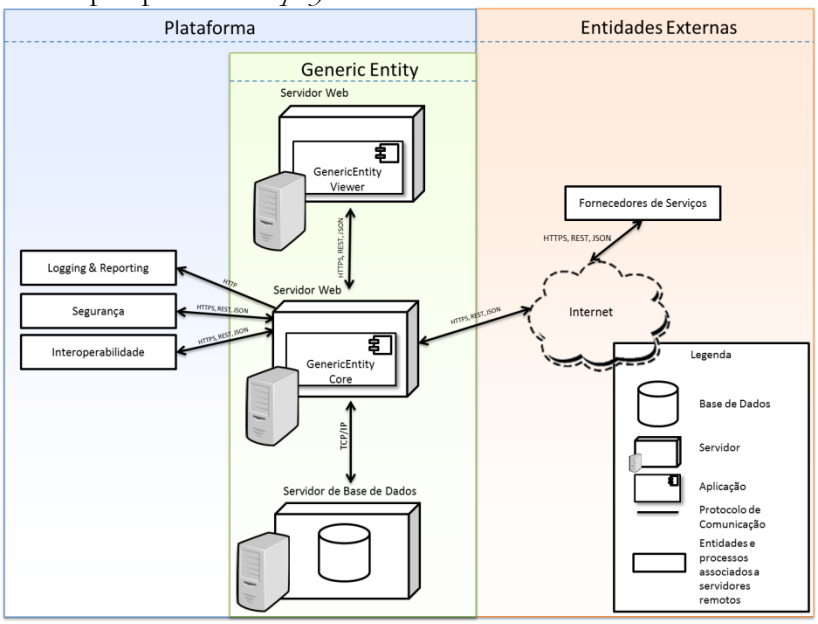

Figura 6 Perspetiva de deployment de nível 0.

A arquitetura é composta pelos seguintes intervenientes:

- **Generic Entity Core –** está encarregue de armazenar os dados clínicos da plataforma e que pode ser acedida através de uma API REST.

[12] REST é um estilo de arquitetura que define um conjunto de restrições que, quando aplicadas à arquitetura de um sistema distribuído, incluem propriedades desejáveis como baixo acoplamento e escalabilidade horizontal [87].

- **Sistema de gestão de base de dados –** com a qual o Generic Entity Core comunica.
- **Generic Entity Viewer –** é a aplicação de visualização dos dados (no caso do PHR dados clínicos) e configuração da aplicação *web* de visualização.
- **Fornecedores de serviços –** que venham a utilizar o Generic Entity Core para aceder aos dados clínicos.
- **Módulo de *logging* e *reporting*** – módulo que se pretende vir a integrar com vários subprojectos para criação de um painel de instrumentos central capaz de monitorizar o estado de toda a plataforma.
- **Módulo de segurança –** que trata da autenticação (verificar a identidade de aplicações que queiram interagir com a plataforma) e autorização (verificação de permissões de acesso aos dados).
- **Módulo de interoperabilidade –** será desenvolvida para facilitar a troca das informações para um grande número de serviços que pretendam receber e enviar a informação num formato específico. Fundamentalmente no contexto do Generic Entity Core o módulo de interoperabilidade irá encapsular e interpretar informação em formatos específicos de comunicação.

5.3.2 Visão de Nível 1

Esta secção pretende documentar os componentes que constituem a aplicação e respetivos fluxos e interações.

5.3.2.1 Perspetiva Estática

A Figura 7 apresenta a perspetiva de alto-nível da vista estática da plataforma com a qual se pretende descrever a organização lógica do sistema, com foco na divisão de responsabilidades e implementação.

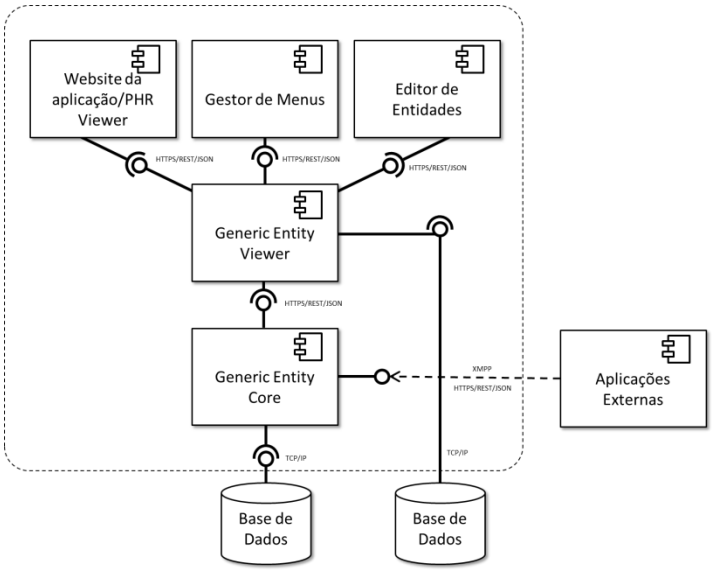

Figura 7 Perspetiva estática de nível 1.

Pela figura é possível constatar que a Generic Entity se divide em cinco grandes componentes:

- **Generic Entity Core –** responsável pelo armazenamento e disponibilização da configuração de entidades e dos dados das entidades ao Viewer. Este módulo possibilita a manipulação de novos dados, ou de dados existentes, ao disponibilizar uma interface genérica para criar, aceder, alterar e eliminar itens de uma entidade, sem a necessidade de alterar a camada de acesso a dados.
- **Generic Entity Viewer –** interpreta a configuração e os valores e cria as vistas sobre os dados. Este módulo é responsável por gerar a interface do utilizador segundo as configurações armazenadas.
- **Gestor de Menus e de Entidades –** possibilita que os administradores do sistema façam gestão de entidades e definam que menus compõe a aplicação *web* (para o caso de uso do TICE é o painel de configuração do PHR Viewer), que componentes – por exemplo: formulários, listas de registos, gráficos dos dados – aparecem em cada menu, como é que os componentes se relacionam – por exemplo editar um registo pode causar a atualização automática de uma lista de registos ou de um gráfico, ou despoletar uma mensagem de alerta.
- **Editor de Entidades –** permite criar novas entidades e editar entidades existentes de forma visual.
- **O *Website* da Aplicação (neste caso é o PHR Viewer) –** é uma renderização das configurações previamente descritas que usa *routing* e *templating* do lado do cliente, em conjunto com pedidos AJAX, para providenciar uma experiência de utilização confortável e rápida sem comprometer a segurança dos dados.

O Gestor de Menus, Editor de Entidades e *Website* da Aplicação são aplicações do lado do cliente criadas utilizando Backbone – uma *framework* de JavaScript.

Juntos, os cinco componentes anteriores permitem mapear um esquema de base de dados numa interface para o utilizador. O sistema reflete automaticamente em todas as suas camadas modificações feitas às definições das entidades genéricas ou dos menus.

5.3.2.2 Perspetiva Dinâmica

A Figura 8 apresenta a perspetiva dinâmica de nível 1 da plataforma:

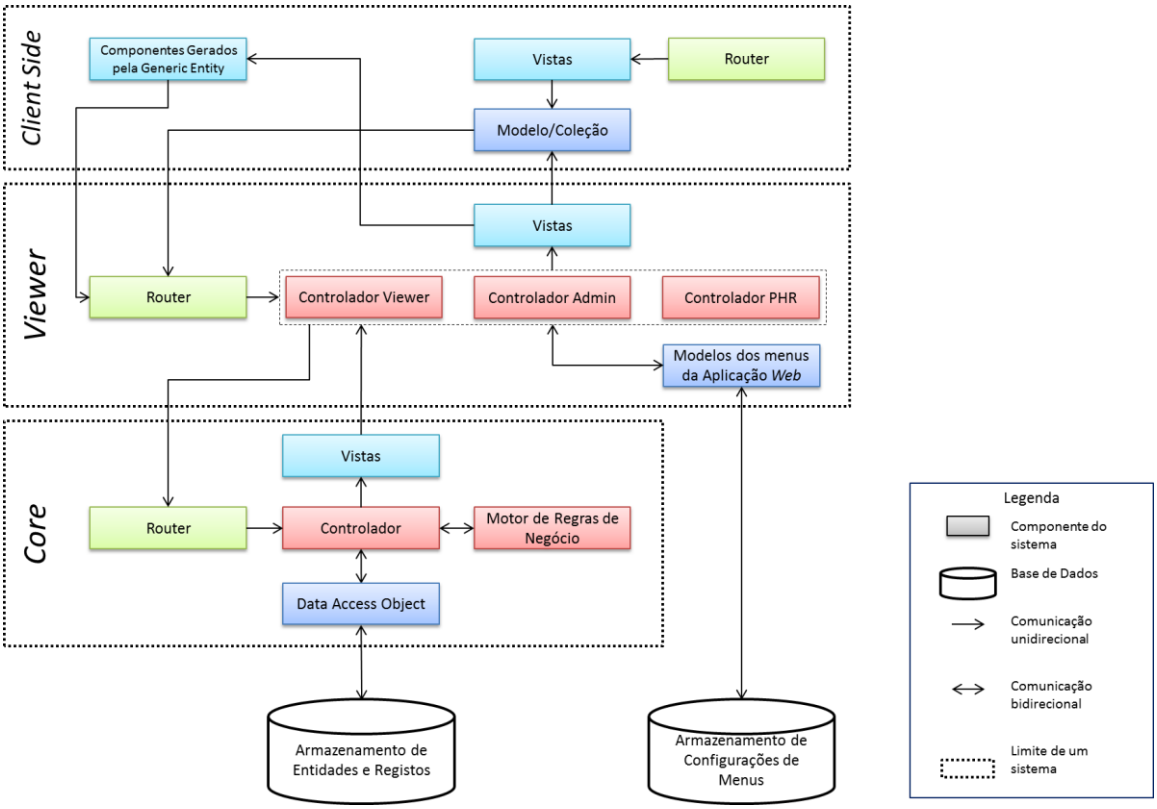

Figura 8 Perspetiva dinâmica de nível 1.

Nesta figura é possível identificar quatro camadas lógicas (distinguidas por cores) que são contempladas no esquema:

- **Camada de apresentação (azul claro) –** onde se encontram os componentes responsáveis pela formatação e apresentação de informação.
- **Lógica de fluxo (verde) –** responsável por traduzir os pedidos *HyperText Transfer Protocol Secure* (HTTPS) para chamadas a ações, utilizando o componente *router* da *framework web* Play e da *framework* Backbone do lado do cliente. Esta lógica existe no lado do cliente que recebe pedidos do *browser*, no Viewer que recebe pedidos do lado do cliente e no Core da Generic Entity que recebe pedidos do Viewer.
- **Lógica de negócio (vermelho) –** inclui as regras para gerir os objetos de negócio. Aqui é importante considerar que as regras de negócio são específicas da Generic Entity e como as entidades genéricas definidas devem ser geridas, e não do PHR. No PHR a lógica de negócio vai estar nas configurações de entidades guardadas na base de dados e no PHR Viewer que é suportado pelo Generic Entity Viewer. Salienta-se ainda o motor de regras de negócio no Core que por cada submissão (POST ou PUT) realizada corre as regras especificadas na configuração, se existirem, e devolve o resultado da execução em JSON.
- **Lógica de acesso a dados (azul escuro) –** existem dois sistemas de persistência distintos, o do Core da Generic Entity onde são guardados os registos e configurações das entidades e o do Viewer onde se guardam as configurações da aplicação de visualização de dados – incluindo os menus, submenus e respetivos elementos que os constituem e que podem ser configurados através do painel de administração. No Core o componente DAO (que será apresentado na secção 5.4.1 Camada de Acesso a Dados) abstrai e encapsula toda a lógica de acesso à fonte de dados e é usado pela camada de lógica de negócio para interagir

com a base de dados. No Viewer é utilizado um modelo ORM para guardar as configurações dos menus.

Pela figura é possível ainda constatar que os componentes gerados pela Generic Entity (formulários e vistas) têm a lógica básica do seu funcionamento autocontida no código gerado e sabem que pedidos realizar ao Viewer para manipularem os dados.

5.3.2.2.1 Suspender Pedidos HTTPS

No seu funcionamento normal a Play destina-se a trabalhar com pedidos muito curtos. A *framework* utiliza uma *pool* de *threads* fixa para processar pedidos colocados em fila pelo conector HTTP. Para conseguir resultados ótimos a *pool* de *threads* deve ser tão pequena quanto possível. O tamanho da *pool* considerado ótimo e definido por defeito pela *framework* em ambiente de produção é o número de processadores + 1 [68].

Isto implica que se o tempo de processamento de um pedido for muito longo (por exemplo esperar por uma computação longa) vai bloquear a *pool* de *threads* e penalizar a capacidade de resposta da aplicação. É claro que é possível adicionar mais *threads* à *pool*, mas tal resultaria em recursos desperdiçados e, note-se, o tamanho da *pool* nunca será infinito logo assumir uma estratégia como ter uma *thread* por utilizador e estimar o número máximo de utilizadores não seria viável.

Para dar resposta a estes casos de uso a Play permite a suspensão temporária de um pedido. O pedido HTTPS mantém-se ligado, mas a execução do pedido é retirada da *pool* de *threads* e tentada novamente mais tarde. É possível definir que o pedido será retomado após um período fixo, ou pode-se esperar que esteja disponível um valor de Promise. Na Generic Entity emprega-se a segunda opção para ir buscar conteúdo de forma assíncrona a URLs remotos. A Play permite tratar destas operações de uma forma muito transparente. Por exemplo, cada chamada ao método play.libs.WS.WSRequest.getAsync() executa um pedido GET assíncrono e retorna um play.libs.F.Promise. O método de ação suspende o(s) pedido(s) HTTPS chamando await(...) na combinação de instâncias de Promise (pode ser um ou mais pedidos). Quando todas as chamadas remotas têm resposta a *thread* retoma o processamento e renderiza a resposta.

5.3.2.3 Perspetiva de Deployment

Na perspetiva de Deployment serão apresentados os principais ficheiros das aplicações e a respetiva árvore de diretórios. A apresentação desta estrutura suportará desenvolvimentos futuros neste projeto. A Figura 9 apresenta a estrutura de diretorias base de um projeto em Play e os principais ficheiros e *packages* da GenericEntityCore e GenericEntityViewer.

Figura 9 Hierarquia de pastas base de um projeto em Play, e das aplicações GenericEntityCore e GenericEntityViewer.

Uma aplicação em Play segue uma estrutura predefinida de diretorias. As mais importantes são:

- **app/** contém o núcleo da aplicação dividido em diretorias para modelos, controladores e vistas. Pode conter outras *packages* Java e é o diretório onde são colocados os ficheiros .java com o código fonte.
- **conf/** contém todos os ficheiros de configuração para a aplicação. Destacam-se os ficheiros: application.conf, que é o principal ficheiro de configuração; o ficheiro de definição de *routes*; e os ficheiros de mensagens usados para internacionalização. Na aplicação Core da Generic Entity este foi o diretório escolhido para colocar o ficheiro .sql que cria a base de dados caso esta não exista quando se executa a aplicação.
- **lib/** contém todas as bibliotecas Java adicionais no formato .jar.
- **public/** contém todos os recursos disponíveis publicamente, que incluem ficheiros JavaScript, *stylesheets* e diretórios de imagens.
- **test/** contém todos os testes da aplicação, tanto os programados como testes Java JUnit como os programados em Selenium.
- **logs/** onde a aplicação armazena os *logs*.

Nas tabelas seguintes são apresentados os principais ficheiros das aplicações GenericEntityCore e GenericEntityViewer:

Tabela 8 Principais ficheiros da aplicação GenericEntityCore.

Ficheiro/*Package*	Descrição
Bootstrap.java	Estende um Bootstrap Job da Play sendo executada quando a aplicação é iniciada. Verifica se a base de dados existe, se não existir cria-a. Também se pode usar para recriar a base de dados.
controllers/Application.java	Tem todos os métodos que representam ações disponibilizadas pela API do Core. Os métodos extraem dados relevantes do pedido HTTP, leem ou fazem atualizações na base de dados e devolvem um resultado envolto numa resposta HTTP.
controllers/SingletonJavaSmackAPI	É uma classe que implementa o *design pattern* Singleton[13] e que lida com questões de ligação e envio de dados para o servidor de XMPP.
controllers/businessLogic/EntityRulesResult.java	Representa o resultado da execução das regras em Drools (permite saber se uma submissão é válida ou teve erros e se teve erros quais foram e permite saber se uma submissão tem alertas e, se sim, quais).
controllers/businessLogic/RulesEngine.java	Uma instância desta classe serve como objeto que encapsula a lógica de acesso a classes do Drools. O seu construtor recebe as regras e usa a classe PackageBuilder para fazer o *parsing* e compilação das mesmas. Possui ainda um método para executar as classes carregadas onde é usada a classe WorkingMemory para declarar o conhecimento (estado corrente do objeto GenEntityItem que contém a informação de uma submissão) que o motor de regras deve utilizar para determinar que consequências deve executar.
controllers/businessLogic/UseRulesEngine.java	Esta classe tem uma propriedade que é uma instância da classe RulesEngine. Tem ainda um método que invoca o método executeRules() da classe RulesEngine. É nesta classe que é inserido o conhecimento (objeto GenEntityItem) na WorkingMemory. É ainda utilizado o método setGlobal() para que o motor de regras tenha referência ao objeto entityRulesResult que não é usado como conhecimento, mas sim para devolver o resultado da execução das regras.
controllers/requests/…	Contém classes que representam parâmetros de pedidos que podem ser feitos à API e respostas da API.
models/customExceptions/…	Esta *package* contém as exceções personalizadas criadas para a aplicação. Foi necessário criar exceções que encapsulam a natureza da fonte de dados para que a camada de acesso a dados possa ser alterada sem que isso implique modificações às camadas superiores
models/fieldConfiguration/enumsForConfiguration/…	Esta *package* contém os enumeradores[14] usados na configuração.

[13] O *design pattern* Singleton tem o propósito de criar uma única instância da classe e providenciar acesso global a esta. Para lidar com a ligação ao servidor XMPP basta uma instância e mais que uma poderia comprometer a estabilidade da aplicação. Neste contexto assumiu-se que a Generic Entity nunca vai transmitir dados para mais do que um servidor XMPP. Se for esse o caso a classe que gere a ligação terá que ser alterada.

[14] Num enumerador (*enum type*) os campos consistem num conjunto fixo de constantes.

models/fieldConfiguration/FieldMappingConfiguration.java	Classe que tem como propriedades as configurações gerais da entidade e contém estruturas para os campos (FormField) e vistas (EntityView) que compõe a entidade.
models/fieldConfiguration/FormField.java	Classe que tem como propriedades as configurações do campo (por exemplo: identificação, coluna na base de dados, validações, entre outras).
models/fieldConfiguration/EntityView.java	Classe que tem como propriedades as configurações da vista (por exemplo: identificação, campos da vista – ViewField, entre outras).
models/fieldConfiguration/ViewField.java	Classe que tem como propriedades as configurações do campo da vista (por exemplo: identificação, se é visível, se é editável, se deve ser usado para ordenação, entre outras).
models/DAOFactory.java	Classe abstrata que produz *Data Access Object* s (DAOs) tais como GenEntityDAO e GenEntityItemDAO.
models/GenEntityDAO.java	Interface que define os métodos DAO para o objeto GenEntity (entidade). Encapsula os mecanismos de acesso necessários para trabalhar com a fonte de dados.
models/GenEntityItemDAO.java	Interface que define os métodos DAO para o objeto GenEntityItem (registo de uma entidade). Encapsula os mecanismos de acesso necessários para trabalhar com a fonte de dados.
models/PostgreSQLDAOFactory.java	Implementação concreta da DAOFactory para PostgreSQL.
models/PostgreSQLGenEntityDAO.java	Implementação dos métodos da interface GenEntityDAO para PostgreSQL.
models/PostgreSQLGenEntityItemDAO.java	Implementação dos métodos da interface GenEntityItemDAO para PostgreSQL.
models/GenEntity.java	Objeto de transferência utilizado como portador de dados de uma entidade.
models/GenEntityItem.java	Objeto de transferência utilizado como portador de dados de um registo de uma entidade.

Naturalmente há classes da GenericEntityCore que também são utilizadas no Viewer como as classes das *packages fieldConfiguration* e *requests* e os objetos de transferência GenEntity e GenEntityItem que são os portadores de dados de entidades e registos respetivamente.

Tabela 9 Principais ficheiros da aplicação GenericEntityViewer.

Ficheiro/*Package*	**Descrição**
Bootstrap.java	Carrega dados iniciais necessários para a base de dados (por exemplo um utilizador administrador).
fastTagsForGeneratingViews.java	Classe com os métodos responsáveis pela geração de código das vistas configuradas. Esta classe estende a classe FastTag da Play que irá ser apresentada numa secção posterior. Utiliza as classes na *package* models/fieldWidgets/ para gerar código específico consoante o tipo de campo.
controllers/OAuthAppBaseAuthentication.java	Tem os métodos que tratam da autenticação na aplicação por OAuth.

controllers/Security.java	Tem os métodos que verificam se um utilizador está autenticado, que colocam a informação de um utilizador na sessão (que é mantida por um *cookie* assinado) se este for autenticado com sucesso, que verificam permissões básicas (por exemplo se é administrador) e tem um método que permite autenticar por nome de utilizador e *password* para ser utilizado em desenvolvimento, se o servidor de OAuth estiver em baixo, ou em projetos fora dos TICE que não utilizem o portal.
controllers/Admin.java	Tem todos os métodos do painel de administração e antes do acesso a qualquer método verifica não só que o utilizador está autenticado como também se é administrador, através de anotações que empregam os métodos da classe Security.
controllers/Application.java	Onde se encontram todos os métodos agnósticos ao projeto que lidam com a troca de informação com o Core da Generic Entity.
controllers/MenusAPI	Tem todos os métodos que representam ações disponibilizadas pela API de menus tanto para configuração no painel de administração como para apresentação na aplicação *web* configurada.
controllers/DynamicPersonalHeathRecord.java	Métodos específicos do projeto PersonalHealthRecord. Salienta-se o método index() que devolve a primeira página do PHR se o utilizador estiver autenticado.
controllers/Proposal.java	Métodos da API de submissão por terceiros de propostas de adição ou alteração de entidades.
models/fieldWidgets/…	Tem as classes responsáveis pela geração de código de componentes específicos como por exemplo: textbox, combobox, radio, tabela de dados, entre outros. Estas classes implementam a interface definida em Widget.java. Adicionalmente existe a classe WidgetValidation que gera as validações para os formulários.
models/Menu.java	Representa um menu ou submenu e engloba as suas propriedades. Pode conter MenuComponents e/ou MenuMultilanguageNames.
models/MenuComponent.java	Tem as propriedades de um componente – pedaço de conteúdo de um menu ou submenu. Essas propriedades identificam o tipo de componente (se é uma vista de entidade definida na GenericEntity, ou conteúdo criado por um administrador, ou por exemplo um *iframe*), como obter esse componente (o seu URL) e propriedades que ligam este componente a outros por mecanismos de *publish/subscribe*.
models/MenuMultilanguageName.java	Tem como propriedades uma chave que representa a língua e o nome do Menu nessa língua e é utilizado pelos mecanismos de internacionalização da Generic Entity (e não pelos da *framework* Play).

views/…	Existem Views para cada um dos controladores definidos (tirando aqueles que possuem apenas métodos de API que devolvem JSON). Estas Views estão divididas por diretórios com os nomes dos controladores e contêm ficheiros .html que incluem HTML e código de apresentação na linguagem Groovy.
/public/images/…	Imagens da aplicação.
/public/stylesheets/…	Stylesheets da aplicação.
/public/js/…	JavaScript da aplicação.
/public/js/models/…	Modelos utilizados pelas aplicações construídas com a *framework* Backbone que são apresentados numa secção posterior.
/public/js/views/…	Views utilizadas pelas aplicações construídas com a *framework* Backbone.
/public/tpl/…	*Templates* utilizados pelas aplicações construídas utilizando Backbone.js

5.4 Desenho da Generic Entity

Para alcançar a extensibilidade e flexibilidade pretendidas as entidades que sejam necessárias são definidas e configuradas em JSON e estas configurações são armazenadas na base de dados.

Estes campos contêm informação como:

- Mapeamento de atributos de entidade para colunas de uma tabela da base de dados.
- Propriedades dos campos (tais como restrições sobre os campos e tipo de dados correspondentes).
- Ações a serem despoletadas por eventos. Por exemplo, o evento de submissão de formulário irá tipicamente provocar a ação de armazenamento dos dados na base de dados.
- Configuração de vistas sobre os dados, como por exemplo listagem, inserção, edição e apagamento. Estas vistas podem ser configuradas para apresentar diferentes atributos das entidades (o que é útil por exemplo para definir permissões diferentes de visualização e manipulação de dados pelos utilizadores).
- Regras do lado do servidor para validar registos submetidos e executar operações quando são satisfeitas determinadas condições (por exemplo envio de alerta quando o valor de um atributo ultrapassa um limite estabelecido).

Existem exemplos de utilização da Generic Entity na secção C.2 Funcionamento da Generic Entity do documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)", e no documento "TICE-GenericEntity Manual de Programador".

5.4.1 Camada de Acesso a Dados

Caracteristicamente uma aplicação em Play segue o padrão arquitetural *model-view-controller* (MVC) aplicado à arquitetura *web*. Este padrão divide a aplicação em camadas separadas:

- A camada **Modelo** é uma representação específica do domínio no qual a aplicação opera que confere "significado" a dados brutos. Esta camada encapsula o acesso a dados.
- A **Vista** faz *render* do modelo para uma forma adequada para interação (os exemplos

mais comuns são os formatos *web* HTML – *HyperText Markup Language*, XML e JSON).

- O **Controlador** responde a eventos (tipicamente ações do utilizador na forma de pedidos HTTP) e processa-os, e pode ainda invocar atualizações ao modelo.

No caso da Play, por defeito, a camada modelo utiliza o padrão *Active Record*, em que uma vista da base de dados é "envolta" numa classe de forma a aceder a dados da fonte servindo-se da técnica *Object Relational Mapping* (ORM) para converter dados entre sistemas incompatíveis. No desenho do Core da Generic Entity optou-se por não utilizar estas técnicas e padrões típicos uma vez que não se adequam à solução pretendida. Um exemplo claro desta questão é que há necessidade de poder mudar o esquema da base de dados relacional e tendo atributos de uma classe mapeados para campos de uma tabela não haveria forma fácil de realizar essa tarefa.

Apesar destas soluções não se adaptarem continua a ser útil ter uma camada que abstraia do acesso a dados pelo que se escolheu implementar o padrão de *software Data Access Object* (DAO), que encapsula os mecanismos de acesso necessários para trabalhar com a fonte de dados. A DAO utiliza o padrão *Transfer Objects*[15] para transporte/troca de dados com a camada superior. Por outras palavras, com este padrão a informação retirada da fonte de dados é colocada num objeto tornando muito mais fácil a manipulação da mesma. Uma vantagem direta da utilização deste padrão é que estes *Transfer Objects* têm anotações sobre os seus atributos para que a sua conversão para a representação JSON, ou o inverso, seja automatizada.

Em conjunto são ainda usados os padrões *Abstract Factory*[16] e *Factory Method*[17] para aumentar a flexibilidade do DAO, permitindo a troca do armazenamento subjacente de uma implementação para outra de forma simples. Isto é, torna-se possível alterar a forma de armazenamento mantendo intactas todas as camadas superiores. Tomando um exemplo em que se pretende implementar esta estratégia para PostgreSQL e MongoDB o diagrama de classes seria o seguinte (Figura 10):

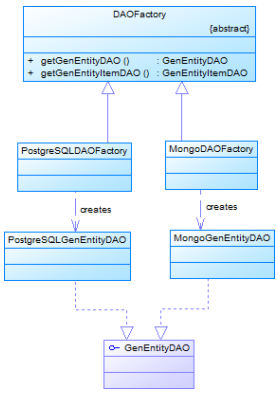

Figura 10 Diagrama de classes exemplificativo do uso da estratégia de combinar os padrões DAO, Abstract Factory e Factory Method.

A classe abstrata DAOFactory produz DAOs tais como GenEntityDAO e GenEntityItemDAO. Esta estratégia usa a implementação *Factory Method* nas *factories* produzidas pela *Abstract Factory*. A estratégia utilizada é descrita mais detalhadamente em

[15] *Transfer Object* é um padrão de *design* usado para transferir dados entre subsistemas de aplicações de *software*.
[16] *Abstract Factory* é um padrão de *design* no qual é providenciada uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar a sua classe concreta [88].
[17] *Factory Method* é um padrão de *design* no qual se define uma interface para criação de um objeto, mas se deixa as subclasses decidir que classe instanciar [88].

[69]. As consequências do desenho da camada de acesso a dados encontram-se no documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)", na secção C.1 Consequências do Desenho da Camada de Acesso a Dados.

5.4.2 Modelo de Dados

De seguida é apresentado o modelo de dados físico da Generic Entity (Figura 11).

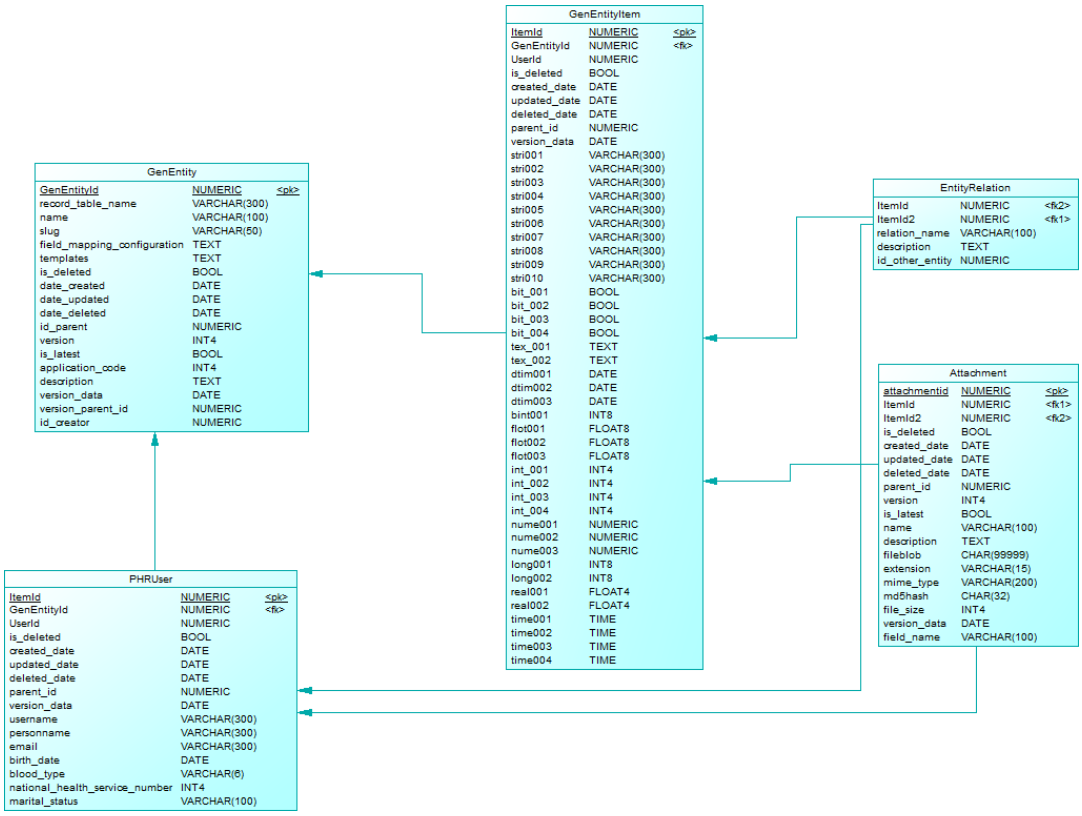

Figura 11 Modelo de dados físico da Generic Entity.

O diagrama da Figura 11 apresenta quatro tabelas do sistema e uma tabela personalizada de exemplo:

- A tabela **GenEntity** guarda a definição da entidade genérica e, para este efeito, possui as colunas field_mapping_configuration – mapeamento de campos da entidade genérica em colunas da tabela GenEntityItem, consoante o tipo de dados – e *templates* – vistas sobre os dados – que se destinam ao armazenamento das configurações em JSON.
- A tabela **GenEntityItem** é uma tabela genérica com colunas pré-criadas para vários tipos de dados. A um registo GenEntity podem corresponder vários registos GenEntityItem com os dados introduzidos na entidade genérica.
- A tabela **Attachment** serve para guardar documentos do sistema e suporta controlo de versões.
- A tabela **EntityRelation** armazena relações entre registos. Por exemplo, uma entidade para armazenamento de historial de medicação de um utente pode ter uma relação para um ou mais registos de uma entidade criada para armazenar medicamentos e os seus detalhes.
- A tabela **PHRUser** é um exemplo de uma tabela personalizada normal que pode ser associada à configuração de uma entidade.

A estratégia de colunas pré-criadas evita que sejam necessários *ALTER TABLES* frequentes para incluir novos campos na tabela. Contudo, estas operações são possíveis caso se esgote o total de colunas de um tipo da tabela genérica e sejam necessárias novas colunas (acontecimento que será extremamente raro na versão real do modelo de dados). Apesar da raridade deste evento é interessante notar que o PostgreSQL é conhecido por ser extremamente rápido na adição e remoção de colunas mesmo em casos em que a tabela esteja populada com milhões de registos. Existem SGBDs em que esta intervenção pode demorar segundos, minutos ou mesmo horas, mas no PostgreSQL a operação é quase imediata [70]. Adicionalmente, caso considerações de desempenho levem a que se pretenda adicionar um índice a uma coluna da tabela genérica quando esta já estiver populada, o PostgreSQL permite criar índices de forma concorrente (sem manter o *lock*) para que não haja tempo de inatividade do sistema por a tabela estar bloqueada [70].

Considera-se assim que o número máximo de atributos de uma entidade genérica é limitado pelo número máximo de colunas de uma tabela em PostgreSQL que é no mínimo 250 e no máximo 1600, dependendo do tipo de colunas. Uma forma de solucionar este problema seria permitir que o preenchimento de dados de uma entidade genérica ocupasse mais que um registo na tabela GenEntityItem, mas tal solução seria complexa e potencialmente prejudicial para o desempenho e simplicidade das *queries*. Considera-se que os campos disponíveis são suficientes e que no caso de uma entidade necessitar de mais seria dividida em duas ou mais entidades ao nível do negócio.

A Figura 12 apresenta o modelo de dados conceptual das configurações armazenadas no formato JSON, na tabela GenEntity.

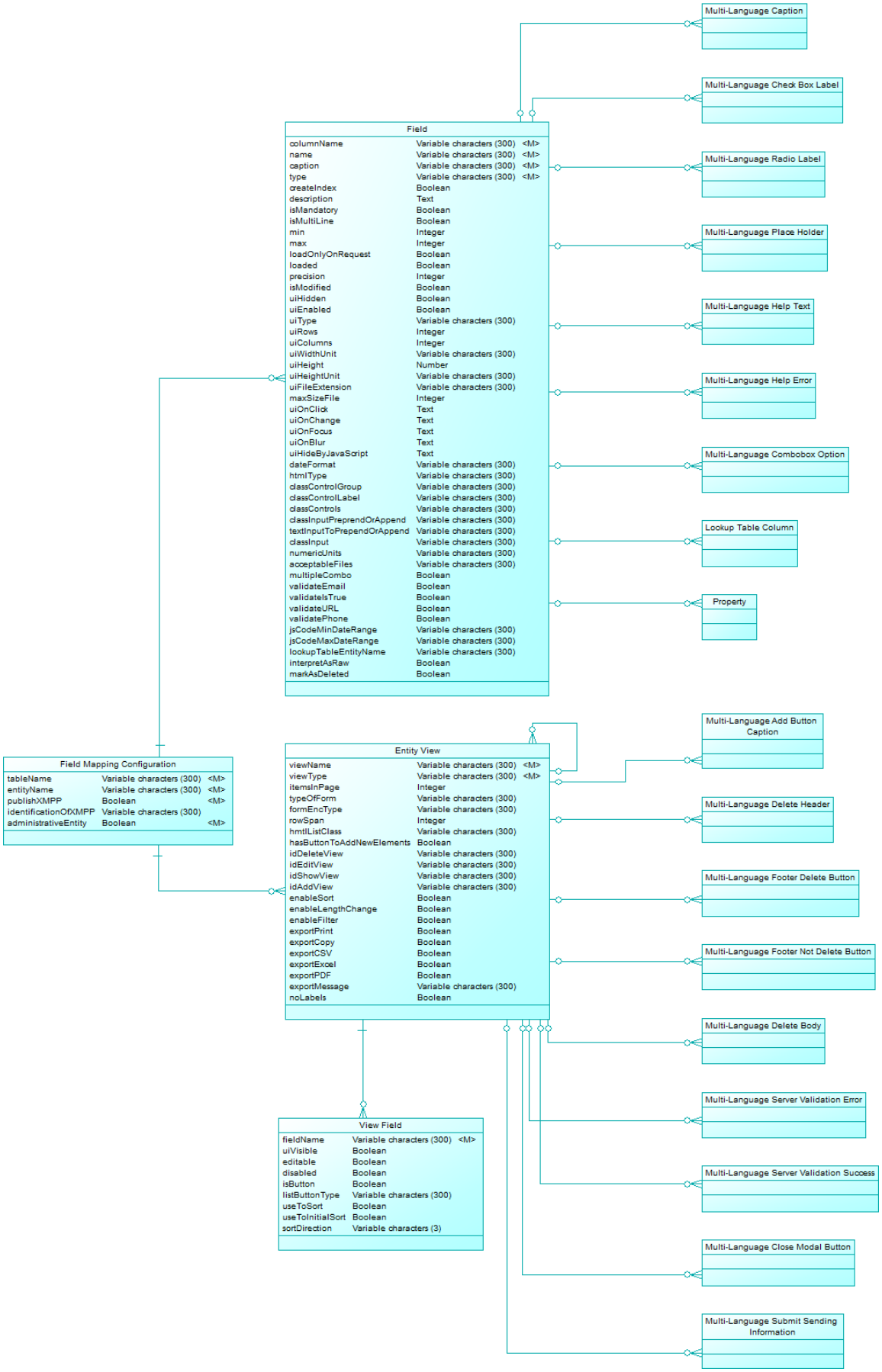

Figura 12 Modelo de dados conceptual da configuração armazenada em JSON.

Apesar de aparentemente simples o modelo físico apresentado na Figura 11 representa uma camada de negócios complexa, uma vez que a configuração armazenada na tabela GenEntity tem o seu próprio modelo de dados que é representado na Figura 12. Como se pode observar no diagrama da Figura 12 o Field Mapping Configuration – que contém os dados genéricos da entidade – encontra-se ligado a Field – que armazena as propriedades de um campo/atributo – e a Entity View – que armazena as propriedades de uma vista. Naturalmente, uma Entity View pode estar ligada a outras Entity Views, para representar, por exemplo, casos de vistas de listagem de registos que podem ter vistas para operações de adição, visualização, apagamento e edição de dados. A Entity View relaciona-se com o View Field – que armazena as propriedades de um campo (se é editável, se deve ser usado para ordenação, entre outras) que faz parte da vista. Existem ainda diversas estruturas ligadas a Field e Entity View que representam, na grande maioria, propriedades em várias línguas. Field tem ainda relação com uma estrutura que identifica as colunas a apresentar caso se trate de um *lookup field*.

5.4.3 Uniform Resource Identifier's do Core da Generic Entity

Os URIs são usados para apontar para recursos válidos de um *web service*. Os URIs escolhidos devem possuir algumas propriedades importantes, como carácter descritivo – o URI deve descrever o conteúdo e exibir uma estrutura previsível. O padrão definido para os URIs da Generic Entity pretende ser tão simples quanto possível e é apresentado abaixo.

URIs da Generic Entity

```
/[domain]/entities
/[domain]/entities/{entityId}
/[domain]/entities/{entityId}/items
/[domain]/entities/{entityId}/items/{itemId}
/[domain]/documents/{documentId}
```

URIs da Generic Entity Personalizados para uma Entidade

```
/[domain]/{entity}
/[domain]/{entity}/items
/[domain]/{entity}/{itemId}
```

Em REST um recurso pode ter um ou mais URIs e um URI designa exatamente um recurso [71]. Neste caso foi escolhido ter mais do que um URI a apontar para um recurso para tornar a API mais conveniente para utilização. Na verdade, os recursos podem ser acedidos tendo em conta a natureza genérica da entidade providenciando o id da entidade, ou podem ser acedidos abstraindo desta característica utilizando os recursos como se estes fossem estáticos. Com efeito, ao ser criada uma entidade a API REST que lhe dá acesso também é automaticamente criada. Por exemplo, supondo que criamos uma entidade "medications" com id=3 e pretendemos aceder ao registo com id=12 desta entidade, é possível utilizar tanto o URI /[domain]/entities/3/items/12 como o URI /[domain]/medications/12. O segundo URI é mais familiar e natural numa API REST e permite que esta seja utilizada mesmo por clientes que desconhecem por completo a natureza genérica desta aplicação.

Ao utilizar os URIs personalizados abstrai-se da natureza genérica e dinâmica da aplicação o que implica que não se podem criar ou atualizar entidades desta forma (apenas se podem atualizar, apagar e criar *items*/registos da entidade).

Existem muitas outras opções na utilização da API para personalizar um pedido feito ao Core. De seguida são apresentados alguns exemplos:

- **GET *[domain]*/entities/?apiRequest.parentID=3&apiRequest.version=2** – Este pedido indica que se pretende obter a entidade que tem como entidade pai a entidade com id=3 e que se encontra na segunda versão.
- **GET *[domain]*/entities?apiRequest.start=3&apiRequest.onlyLatest=false** – Devolve as entidades começando na terceira encontrada (por defeito são devolvidas com ordenação pelo *timestamp* de criação), e não apenas as versões em vigor (também versões passadas das entidades).
- **PUT *[domain]*/entities/12&apiRequest.save=forceUpdate** – o parâmetro forceUpdate atualiza as informações sem criar nova versão da entidade mesmo que a configuração JSON tenha sido alterada. Por defeito, sem utilizar este parâmetro, é criada uma nova versão da entidade se for detetado que a sua configuração mudou. No corpo de um pedido HTTP POST ou PUT são colocadas as chaves (nomes dos atributos) e respetivos valores. No PUT só se colocam os que se pretende atualizar assumindo-se que todos os outros atributos se mantêm.
- **GET [domain]/EntityPersonalHealthData/items/?apiRequest.columnsAndValues.username=Pedro& apiRequest.columnsAndValues.age=3** – columnsAndValues é um mapa especificado quando se quer filtrar por coluna com determinado valor. É ainda possível escolher se essa comparação requer correspondência perfeita ou se, pelo contrário, basta que o valor esteja contido (quando se trata de uma *string*). Por defeito está a *true* porque correspondências perfeitas apresentam melhor desempenho.

Para uma descrição completa da API do Core da Generic Entity, incluindo os parâmetros que podem ser utilizados para obtenção seletiva de registos (ou de atributos dentro de registos) e entidades, paginação, tipos de atualização de entidades (por exemplo se será criada nova versão ou se irá ser substituída a antiga) e respostas do servidor em cada caso, deve consultar o anexo "TICE-GenericEntity Manual do Programador". Este inclui ainda um enquadramento das escolhas feitas face aos princípios REST.

5.5 Gerador de Código

Existem dois tipos principais de geradores de código [72]:

- **Geradores de código passivos** são executados para produzir um resultado. A partir desse momento o resultado torna-se autónomo e a sua origem torna-se irrelevante. Geradores de código passivos poupam trabalho de programação. São basicamente *templates* parametrizados gerando determinado *output* a partir de um *input*.
- **Geradores de códigos ativos** são usados cada vez que o seu resultado é necessário. Este resultado é descartável, já que pode ser sempre reproduzido pelo gerador de código.

Caracteristicamente, os geradores de código ativos, como é o caso do gerador da Generic Entity, leem e interpretam alguma forma de *script* ou ficheiro de controlo com as configurações e produzem o seu resultado. Neste caso, como pode ser observado na Figura 13, o resultado gerado é uma API para manusear a entidade e formulários e vistas para manipulação dos dados.

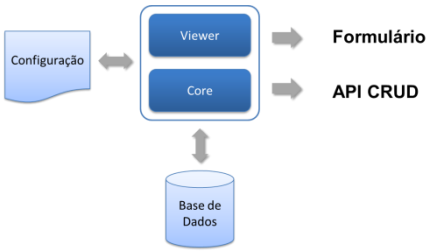

Figura 13 Esquema de geração de código da Generic Entity.

Enquanto os geradores de código passivos são uma simples conveniência, os geradores de código ativos são uma necessidade fundamental na aplicação do princípio de *software Don't Repeat Yourself* (DRY) [18]. Com um gerador de código ativo, é possível pegar numa única representação de um pedaço de informação e convertê-la em todas as formas de que a aplicação necessita. Isto não é duplicação, uma vez que as formas derivadas são descartáveis, e são produzidas pelo gerador de código à medida que são necessárias.

5.5.1 Gerador de Vistas de Alto Desempenho

Uma *tag* é utilizada para manter tarefas repetitivas do *template* curtas. Adicionalmente, dado que tudo o que é necessário saber para o seu uso é HTML e a linguagem de *templating* embutida (neste caso Groovy), mesmo os programadores *web* sem conhecimento do *backend* podem aproveitar esta solução com facilidade [73].

Ao desenhar o gerador de vistas do Viewer foi considerado que este, sendo um componente central da aplicação, devia apresentar-se como uma *tag* de alto desempenho. Como tal, foi utilizado um mecanismo interno da Play, as FastTags, que acelera a execução. Com efeito, a maioria das *tags* nativas suportadas pela Play são FastTags de forma a manter o desempenho elevado. As *tags* criadas com este mecanismo são compiladas, já que são totalmente programadas em Java.

Assim, ao utilizar uma FastTag para a geração de vistas: garante-se uma maior simplicidade e organização do código, consegue-se um desempenho superior e mantém-se o gerador agnóstico à linguagem de *templating* utilizada pela Play (o Groovy para além de ser substituível pode mesmo deixar de ser a principal e mais vantajosa linguagem de *templating* em futuras versões da *framework*).

5.6 Aplicações Client-Side

O Backbone é uma *framework* arquitetural leve que permite a criação de aplicações JavaScript não triviais utilizando uma organização e estrutura do estilo MVC. O Backbone não é verdadeiramente considerado MVC, o C é de Collection e não de Controller, mas ainda assim oferece muitos dos mesmos benefícios e permite escrever código poderoso e sustentável. A Figura 14 apresenta os blocos de construção de aplicações disponibilizados pelo Backbone.

[18] De acordo com o princípio DRY cada pedaço de conhecimento só pode ter uma única representação oficial dentro de um sistema e esta não deve ser ambígua [72]. Este princípio visa manter o desenvolvimento de *software* fiável e tornar os sistemas mais fáceis de perceber e manter.

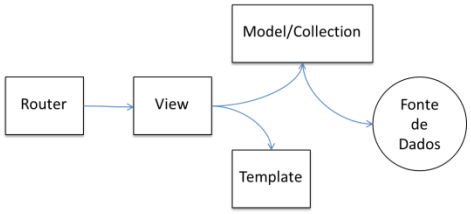

Figura 14 Esquema de blocos de construção de aplicações disponibilizadas pelo Backbone.

O Backbone dá estrutura a aplicações *web* providenciando modelos com *binding* de chave-valor e eventos personalizados, coleções com uma API rica de funções de enumeração, vistas com manipulação de eventos declarativa e pode ligar tudo à API do servidor através de uma interface JSON RESTful. O componente *router* mapeia URLs para funções e permite histórico e *bookmarking* [74].

A escolha desta *framework* decorreu do requisito organizacional de manter as tecnologias tão homogéneas quanto possível. Na realidade, o Backbone já é usado no portal e esta escolha foi fruto de uma análise competente do estado da arte de *frameworks client-side*. O Backbone adequa-se bem às necessidades deste projeto no qual se pretendiam criar aplicações interativas, complexas (pesadas no *frontend*) e orientadas a dados.

Para o *templating* utilizaram-se as capacidades do Underscore.js que é uma biblioteca utilitária que fornece funcionalidades melhoradas ao JavaScript e da qual o Backbone depende.

Neste projeto o Backbone foi utilizado para criar três aplicações distintas que utilizam as capacidades da *framework* de forma diferente consoante as necessidades.

5.6.1 Aplicação Viewer

Neste caso a *framework* foi utilizada para construir uma aplicação grande de uma única página, sem adicionar nada mais ao ficheiro HTML, apenas pela manipulação do conteúdo do *body* utilizando JavaScript para carregar os *templates* HTML e o seu teor. Esta aplicação é gerada através das configurações de menus, submenus, componentes e entidades.

A aplicação é composta por:

- **TabMenuModel** – é a entidade singular que representa cada menu e providencia uma forma de ler atributos arbitrários de um conjunto de dados.
- **MenuColletion** – é essencialmente uma coleção de modelos do tipo TabMenuModel.
- **TabMenuListView** – é uma View. A aplicação itera pela coleção e instancia um TabMenuListItemView para cada menu e adiciona à lista de menus.
- **TabMenuListItemView** – é uma View que contém uma função que funde os dados de um modelo com o *template* tab-menu-list-item (que apresenta os menus).
- **AppRouter** – providencia o ponto de entrada para a aplicação através de um conjunto de URLs *deep-linkable*. São definidos três *routes*:
    - O *route* por defeito (“”) que apresenta a lista de menus e submenus.
    - O *route* “menus/:id” apresenta os detalhes de um menu específico (um menu sem menu pai) que são o seu conteúdo (os componentes que contém) e os seus submenus (menus filhos).
    - O *route* “menus/:id/:subid” apresenta os detalhes de um submenu específico

(os componentes pelos quais é formado).

O AppRouter é usado como sistema de navegação da aplicação. Cada vez que um utilizador clica num *link*, o segmento de *hash* do URL vai mudar despoletando o Backbone para navegar para a vista correta. Como já foi referido os URLs são *deep-linkable*, o que significa que a aplicação mantém continuamente o seu URL sincronizado com o seu estado atual. Isto permite copiar o URL da barra de endereços em qualquer momento e reutilizá-lo ou partilhá-lo para voltar exatamente ao mesmo estado.

Cada menu e submenu podem ter componentes adicionados através do painel de administração. Estes componentes podem ser conteúdo gerado e formatado por um administrador (através de um *rich-text editor*[19]) ou podem ainda ser componentes configurados por um administrador e gerados pela Generic Entity (vistas e formulários de visualização, edição, adição e listagem de dados). Quando se clica num menu ou submenu os seus componentes são carregados para a página. Os componentes que são gerados pela Generic Entity são autónomos, isto é, já têm código gerado que os administra (valida formulários, trata das ações e eventos, atualiza o seu conteúdo e gere as submissões).

5.6.1.1 Ligação dos Componentes do Viewer

Ao criar a aplicação Viewer tornou-se aparente a necessidade de ligar os componentes com algum tipo de relação para tornar a aplicação mais interativa e não ser necessário ao utilizador fazer *refresh* à página para visualizar as consequências óbvias das suas ações. Pode-se tomar como exemplo a adição de uma nova medida de frequência cardíaca em que o utilizador espera ver não só a tabela de listagem a atualizar automaticamente (comportamento que já é gerido pelo código gerado pela Generic Entity), mas também outros componentes como o gráfico de visualização dos dados e o contador de alertas (caso o novo registo tenha despoletado um alerta de saúde). Para alcançar este objetivo foi utilizado o *design pattern* Observer, também conhecido como *publish/subscribe* (Pubsub). Essencialmente, o Pubsub permite a subscrição de eventos. Temos, de um lado, componentes à escuta desses eventos e, do outro, componentes a publicar eventos. Para cada componente no painel de administração é possível especificar uma lista de eventos que publica e outra lista de eventos que subscreve. Para os componentes gerados pela Generic Entity o que acontece já está previamente definido: se um componente tem uma lista de eventos a publicar, vai fazê-lo quando houver algum tipo de manipulação dos dados (inserir, editar ou apagar um registo), se um componente subscreve a uma lista de eventos vai atualizar os seus dados sempre que for notificado para isso. Para os componentes que não são gerados pela Generic Entity é possível adotar este comportamento ou programar um completamente diferente. Para o exemplo anterior o gráfico de frequência cardíaca e contador de alertas subscrevem os eventos reloadHeartRate e alertsReload, respetivamente, e as funções que os subscrevem recarregam os valores sempre que são notificadas. Por sua vez, a tabela de frequência cardíaca publica os dois eventos referidos sempre que os seus dados são manipulados.

5.6.2 Configurador da Aplicação de Visualização

No painel de administração é possível configurar que menus, submenus e componentes vão

[19] *Rich-text editors* [89] são componentes *web* que permitem ao utilizador editar e inserir texto usando um *browser*. São essencialmente editores WYSIWYG ("*what you see is what you get*") baseados na *web*. Neste caso o *rich-text editor* é utilizado como forma de permitir a inserção de conteúdo gerado e formatado por um administrador.

compor a aplicação de visualização. Mais uma vez esta é uma aplicação feita em Backbone. Ao contrário da anterior que só lê dados dos modelos esta utiliza mais ativamente a integração do Backbone com serviços REST para providenciar operações CRUD sobre os menus e submenus. Quando os dados do *backend* são expostos através de uma API RESTful pura (como é o caso), torna-se extremamente fácil obter (GET), criar (POST), atualizar (PUT) e apagar (DELETE) modelos usando a API Model do Backbone. Ao fazer *bind* das Views (listagem e detalhes) aos modelos quando os dados de um modelo mudam todas as vistas ligadas ao modelo voltam a renderizar sem ser necessário código complexo para sincronizar a interface.

Os serviços do *backend* são expostos da seguinte forma:

Tabela 10 Serviços de menus disponibilizados pelo backend.

Método HTTP	URL	Ação
GET	/api/menus	Obter todos os menus
GET	/api/menus/{<[0-9]+>id}	Obter menu com id=={id}
POTS	/api/menus	Criar novo menu
PUT	/api/menus/{<[0-9]+>id}	Atualizar menu com id=={id}
DELETE	/api/menus/{<[0-9]+>id}	Apagar menu com id=={id}

Esta aplicação é composta por blocos (Model, Collection, View, Router) idênticos à aplicação Viewer, tendo um adicional, a View de detalhes e edição que funde os dados de um modelo a um *template* que define um formulário para visualizar e editar os atributos de um menu.

O modelo de dados do configurador de menus é o seguinte (Figura 15):

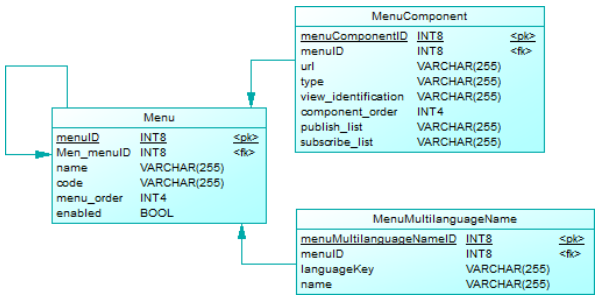

Figura 15 Modelo de dados físico do configurador de menus.

Como se pode observar no diagrama um Menu pode ter filhos (submenus), tem um código de identificação, tem um inteiro para indicar a ordem e um atributo a indicar se o Menu está publicado (*enabled*) na aplicação de visualização. Um menu tem ainda MenuComponents com atributos para URL do componente para o obter do *backend*, tipo de componente (listagem, inserção, edição, visualização, entre outros), identificação da vista se for um componente gerado a partir de uma configuração, um inteiro para indicar a ordem, uma lista de eventos a publicar e uma lista de eventos a subscrever. O MenuMultilanguageName contém apenas uma chave a identificar a língua e o nome do menu nessa língua.

5.6.3 Editor de Entidades

Inicialmente a geração de configurações era realizada utilizando a aplicação desenvolvida FormConfigurationBuilder que é descrita no documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)", na secção C.2.1 Configuração de Uma

Entidade Genérica Utilizando o FomConfigurationBuilder.

De forma a facilitar este processo foi desenvolvida uma aplicação que permite criar ou alterar uma entidade da Generic Entity de forma visual. Nesta aplicação não é utilizada a integração REST do Backbone. No início a aplicação recebe a configuração da entidade que pode ser vazia se for uma nova entidade e apenas quando se salva o trabalho realizado é que a configuração é persistida no servidor. Durante o processo de criação ou edição da entidade as configurações só existem em JSON do lado do cliente. Optou-se por não integrar cada ação com um serviço REST (por exemplo cada atributo adicionado ter um serviço exposto no *backend*) porque tal originaria vários pedidos desnecessários ao servidor e aumentaria a latência da aplicação. Esta abordagem é muito mais limpa, mas não tendo implementado persistência do lado do cliente é possível perder o trabalho realizado ao sair da aplicação sem antes o salvar.

Os elementos desta aplicação são os seguintes:

- **FieldModel** – é um modelo com todos os atributos que um campo de uma entidade da Generic Entity pode ter (muitos deles com valores por defeito).
- **FieldCollection** – coleção de FieldModels.
- **DirectoryView** – é uma View mestre que percorre os elementos da coleção e instancia para cada modelo um FieldView. Para além disso tem funções para adicionar campos arrastados (no editor novos campos são adicionados por *drag-and-drop*) e para adicionar novo campo por duplicação de um campo existente.
- **FieldView** – é uma View de um elemento individual que funde os dados do modelo a um *template* de detalhes e tem eventos definidos para renderizar um novo *template* que permite visualizar e alterar as propriedades e salvar alterações.

Uma vez que os campos e as suas formatações estão sempre visíveis – o editor permite pré-visualizar a entidade formatada como um formulário de inserção com todos os seus campos – não é utilizado um componente *router* para navegação como nas aplicações anteriores. De facto, o *router* é excelente para fazer roteamento de pedidos para as localizações importantes/*stateful*/*bookmarkable*, mas neste cenário não há nenhuma localização dessa natureza. Assim, para interagir com a aplicação são utilizados eventos para despoletar ações. Por exemplo cada campo tem um botão de edição de propriedades que faz despoletar uma função que substitui o *template* do campo por um *template* que, para além do campo, apresenta as suas propriedades num formulário que pode ser salvo. Existem ainda eventos definidos para clique no botão de duplicar campo, apagar campo, cancelar edição de campo e salvar entidade.

Este editor é apenas uma primeira versão que ainda não permite editar de forma visual vistas ou regras personalizadas. Assim, para uma nova entidade são criadas vistas por defeito de todos os tipos (inserção, edição, apresentação e listagem) que assumem permissões completas de visualização e manipulação de dados. Ao editar uma entidade com o visualizador pode-se escolher recriar estas vistas por defeito para contemplar novos campos que tenham sido adicionados ou remover campos que tenham sido apagados. Se for escolhido salvar edições a uma entidade sem recriar as vistas vão se manter as vistas antigas só se pondendo usufruir de algumas alterações ao nível da manipulação de dados pela API do Core da Generic Entity. De momento, para utilizar as capacidades de configuração mais avançadas tem que se recorrer ao programa FormConfigurationBuilder, instanciar as classes necessárias e no fim utilizar o *parser* para converter a configuração para JSON. Por fim pode-se recorrer ao painel de administração para submeter a configuração criada (embora incomportável para um utilizador comum, este processo é trivial para um *developer* e mantém

as vantagens de flexibilidade, facilidade e poupança de tempo).

5.7 Asynchronous Module Definition

Utilizando *Asynchronous Module Definition* (AMD) todo o código é separado em módulos. O AMD cria um padrão *standard* para escrever estes módulos de forma a ser possível carregar o código de forma assíncrona.

Usar a *tag script* bloqueia a página, enquanto carrega até que o *Document Object Model* (DOM) esteja pronto. Usando algo como o mecanismo AMD permite ao DOM continuar a carregar, enquanto os *scripts* também estão a carregar. Essencialmente cada módulo está no seu próprio ficheiro, e há um ficheiro que inicia o processo. A implementação mais popular de AMD é o RequireJS [75] e é esta biblioteca que é utilizada neste projeto [76].

O ficheiro mainRequire.js é onde o processo começa sendo executada a função "require" que recebe como argumento um *array* de dependências (os módulos que são necessários para a aplicação). Há medida que acabam de carregar, o que quer que o módulo retorne é passado para uma função de *callback*. Estes módulos iniciais podem ter outras dependências que são carregadas se ainda não tiverem sido (por exemplo a aplicação editor pode carregar as vistas e modelos de que necessita que estão separadas em ficheiros/módulos para facilitar a manutenção da aplicação).

O Require.js [75] tem ainda um optimizador para ser utilizado quando se passa para ambiente de produção, que combina *scripts* relacionados e os minifica, utilizando o UglifyJS, e optimiza o CSS.

5.8 Integração com o Projeto Sensor Care

Extensible Messaging and Presence Protocol (XMPP) (anteriormente conhecido como Jabber) [77] é um protocolo aberto, extensível, baseado em XML, para sistemas de mensagens instantâneas. É possível configurar a Generic Entity para retransmitir registos recebidos pela API (por POST e PUT) por XMPP. Este mecanismo permite a uma aplicação persistir dados na Generic Entity e ao mesmo tempo outra aplicação receber a evolução desses dados em tempo real. Este processo foi implementado especificamente para integração com o projeto Sensor Care, que, em traços gerais, se foca na monitorização de utentes através de sensores que transmitem os dados por Bluetooth para uma aplicação Android. Esses dados são persistidos na Generic Entity e retransmitidos para um servidor XMPP onde está à escuta uma aplicação que trata da apresentação de dados em tempo real para clientes que utilizem um *browser*.

A integração com o projeto Sensor Care é um bom exemplo da simplicidade de utilização da Generic Entity. Antes de mais foi necessário criar uma entidade para armazenar os dados dos sensores de acordo com a especificação do *developer* e configurada para retransmitir os dados submetidos para um canal XMPP definido. Essa entidade com 9 campos demorou menos de 15 minutos a ser criada programticamente (caso tivesse sido criada com o editor teria sido uma tarefa bem mais rápida, mas nessa altura esta aplicação ainda não estava finalizada), incluindo a configuração das datas para um formato específico e a configuração de vistas que permitiam monitorizar os dados de forma visual e adicionar, apagar e editar dados caso fosse necessário para testes. Durante o desenvolvimento do Sensor Care o seu responsável, o estagiário Nuno Rebelo, pediu para alterar o número de campos e nomes devido a diferenças entre os dados que o sensor realmente debitava e os esperados. Essas alterações simples demoraram menos de 1 minuto e foram aplicadas

diretamente em ambiente de produção, enquanto num cenário típico teria que se alterar o modelo de dados programaticamente, aplicar as alerações à base de dados, alterar os mecanismos de visualização dos dados e fazer *redeploy* da nova versão.

A visualização de dados em tempo real implementada pelo meu colega poderá vir a ser integrada no *Personal Health Record*, seja juntando parte dos dois projetos seja por adição de uma *iframe* para a aplicação Sensor Care, como componente de um dos menus do PHR (mecanismo que já está implementado para o caso de ser necessário). Do projeto Sensor Care estava também planeada a criação de um visualizador de histórico das medições por sensores (por tabela e por gráfico), mas essa tarefa foi fechada por se ter tornado redundante, uma vez que a Generic Entity já cria as tabelas de listagem e tem uma API simples e genérica para facilmente criar os respetivos gráficos dos dados persistidos, constituindo uma forma simples e completa de visualizar o histórico. Através da implementação de regras em Drools é ainda possível definir alertas para valores críticos de forma fácil e flexível, que podem ser despelotados mal os dados cheguem ao Core da Generic Entity.

Esta integração foi concluída com sucesso e a Figura 16 apresenta um esquema de alto nível da relação entre os dois projetos.

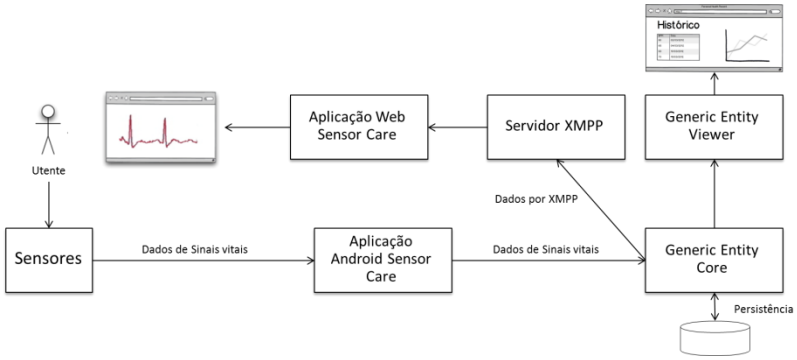

Figura 16 Esquema de alto nível da relação entre a Generic Entity e o Sensor Care.

Atualmente a aplicação em Android utiliza a Generic Entity para persistir os dados e para os retransmitir para serem usados pela aplicação *Web* do Sensor Care, houve uma enorme poupança de tempo e usufrui-se da flexibilidade face a mudanças de requisitos.

5.9 Análise de Segurança

Durante este projecto foi realizada uma primeira análise de segurança que se encontra no documento "TICE-GenericEntity Análise de Segurança". Foram tomados como ponto de partida os 10 riscos de segurança mais críticos em aplicações *web*, identificados pelo *Open Web Application Security Project* (OWASP)[20] [78], mas foram também analisadas outras questões consideradas fundamentais. Nesta secção é feito um resumo da análise de segurança realizada.

5.9.1 Riscos Críticos de Segurança

A1 SQL *injection* – A Generic Entity não utiliza o Model da Play, logo é da sua responsabilidade proteger contra SQL *injection*. Com este objetivo todas as *queries* são

[20] O OWASP é uma comunidade aberta dedicada a promover nas organizações o desenvolvimento, compra e manutenção de aplicações de confiança.

parametrizadas e nunca é utilizada concatenação de *strings* para criar SQL dinâmico. Na Generic Entity são utilizados marcadores de posição e *binding* em conjunto com *prepared statements*[21], portanto os métodos podem ser utilizados sem perigo de SQL *injection*. De facto, os *prepared statements* são resistentes contra SQL *injection*, pois os valores passados como parâmetros são sanitizados (*escaped*) pelo *driver Java Database Connectivity* (JDBC). Isto é, se o *template statement* original utilizado nas *queries* não for criado através de dados externos, não pode ocorrer SQL *injection*. Sendo assim o uso da Generic Entity está protegido contra este tipo de risco.

A2 *Cross-Site Scripting* (XSS) – Ataques XSS ocorrem quando a aplicação envia dados não seguros para o *browser* sem realizar validação ou *escaping* adequados. O motor de *templates* da Play faz *escape* automático de *strings*. Em algumas situações não se pode contar com os mecanismos automáticos da Play para lidar com esta questão; por exemplo, foi necessário ter em conta o *escape* na renderização de conteúdo por *templates* e pelas tabelas de listagem de dados do lado do cliente. Na Generic Entity as entidades estão configuradas por defeito para validar e fazer *escape* do conteúdo apresentado. É possível contornar este mecanismo configurando um ou mais campos da entidade para serem interpretados sem serem *escaped*. Tal é útil para incluir conteúdo como HTML e JavaScript na aplicação de visualização, que seja gerado por um administrador ou pelo próprio sistema.

A3 Falha de Autenticação e Gestão de Sessão – A Play disponibiliza um módulo de segurança que ajuda a implementar um sistema básico de gestão de autenticação e autorização, que a Generic Entity utiliza para intercetar todas as ações e garantir que o utilizador está autenticado e autorizado.

Para precaver este risco os primeiros recursos a proteger são as credenciais e ids de sessão. Existe um princípio, que é salientado na página de boas práticas de segurança do Twitter, que afirma que as aplicações não devem armazenar as *passwords* dos utilizadores [79]. O Generic Entity Viewer não guarda, nem requer, as *passwords* de utilizadores do portal e também não guarda os *tokens* OAuth para comunicar com o portal (que lhe são enviados cada vez que um utilizador do portal entra na aplicação PHR). Como tal, vulnerabilidades que se prendem com a facilidade de adivinhar as credenciais ou sobreposição das mesmas através de funções fracas de gestão de contas (como criação de conta, mudança de *password*, recuperação de *password*, entre outras) dependem exclusivamente dos mecanismos do próprio portal.

Tanto no portal como no *Personal Health Record* construído através da Generic Entity os ids da sessão não são expostos no URL, existem *time-outs* da sessão quando o *browser* é fechado e quando o tempo do *cookie* de sessão ultrapassa o valor definido no parâmetro maxAge e existe o mecanismo de *logout*. Adicionalmente, informação como *tokens*, ids de sessão e credenciais serão apenas transmitidos sobre ligações seguras.

A4 Referência Direta Insegura a um Objeto – Atualmente a Generic Entity não utiliza o mecanismo de acesso a documentos por estratégias de URL descartável, mas nada impede que tais estratégias sejam implementadas no futuro, se forem considerados mais-valias. A Generic Entity irá verificar todos os acessos, isto é, cada uso de uma referência direta a um objeto de uma fonte externa terá que incluir um controlo de acesso para garantir que a entidade está autorizada para tomar a ação sobre o objeto em causa. Esta verificação está dependente da integração com o módulo de permissões que faz gestão das autorizações e com o servidor de OAuth, logo, de momento, ainda não está em funcionamento.

[21] O uso de *prepared statements* com *bind* de variáveis para além de defender contra ataques de SQL *injection* melhora o desempenho.

A5 *Cross-Site Request Forgery* (CSRF) – Um ataque CSRF força o *browser* de uma vítima autenticada a enviar um pedido HTTP forjado, incluindo o *cookie* de sessão da vítima e qualquer outra informação de autenticação incluída de forma automática, para uma aplicação *web*. Isto permite forçar o *browser* da vítima a gerar pedidos que a aplicação vulnerável acredita serem legítimos. Para prevenir este ataque, a primeira preocupação a ter é usar os métodos GET e POST de forma apropriada. Tal significa que apenas o método POST deve ser utilizado para mudar o estado da aplicação. Para pedidos POST a única forma de garantir devidamente a segurança de ações críticas é emitir um *token* de autenticidade. Convenientemente a Play tem um *helper* embutido para lidar com esta ação. Para tal é utilizado o método checkAutenticity(), disponibilizado aos controladores, que verifica se o *token* de autenticidade está presente nos parâmetros do pedido e é válido e envia uma resposta "forbidden" se algo estiver errado. O session.getAuthenticityToken() gera o *token* de autenticidade que apenas é válido para a sessão corrente. A *tag* #{authenticityToken /} cria um campo escondido que pode ser adicionado a qualquer formulário. A Generic Entity protege-se deste ataque fazendo uso do *helper* da Play que segue os princípios fundamentais apresentados pelo OWASP.

A6 Má Configuração da Segurança – Boa segurança requer ter uma configuração segura definida para a aplicação, *frameworks,* servidor aplicacional, servidor *web,* servidor de base de dados e plataforma. Todas estas configurações devem ser analisadas uma vez que muitos componentes não são expedidos com predefinições seguras. Este risco também exige que se mantenha todo o *software* atualizado, abrangendo as bibliotecas de código usadas pela aplicação. Ainda não foi feita uma inspeção para verificar a totalidade destas questões, mas já foram tidas as precauções básicas.

O *software* utilizado neste projeto é, neste momento, o mais recente. Ao fazer *deploy* da Generic Entity já se configurou um ambiente com as características mais importantes (existe um guia da Play para configurar a aplicação para produção) que é ativado ao definir o id para correr as aplicações como "prod", que é o ambiente configurado para produção.

O mecanismo de gestão de dependências da Play já permite definir que se pretende a instalação mais recente dos módulos, portanto é possível realizar atualizações bastando executar um comando. As contas por defeito das tecnologias utilizadas (por exemplo a conta por defeito do PostgreSQL) estão desligadas ou têm nova palavra-passe. Adicionalmente, os mecanismos para lidar com erros estão configurados de forma a impedir *stack traces* e outras mensagens de erro exageradamente informativas.

A7 Armazenamento Criptográfico Inseguro – Neste contexto o mais provável é que a base de dados venha a ser cifrada (no mínimo documentos, determinados campos e ligações entre tabelas), com uma chave pública para a qual apenas a aplicação Core dispõe da respetiva chave privada (protegendo assim também de ataques internos ao excluir os administradores de sistemas). Atualmente a base de dados não é encriptada, mas o PostreSQL tem essas funcionalidades. Quando for decidido ativar este mecanismo existem diversas questões que não podem ser esquecidas como: assegurar o uso de algoritmos apropriados, fortes e *standard*; garantir que *backups* externos estão encriptados, mas que as chaves e respetivas cópias de segurança são geridas separadamente; considerar as principais ameaças das quais se pretende proteger os dados para que possam ser definidas estratégias adequadas; entre outras.

A8 Falhas na Restrição de Acesso a URLs – A Play permite que se definam funções para intercetar chamadas que fazem verificações antes de permitir o acesso e, desta forma, facilita a proteção dos recursos e torna menos provável que algum *route* fique desprotegido. As permissões de acesso aos dados vão ser geridas por um módulo de

permissões genérico e será necessário verificar com este a autorização em cada acesso. Os mecanismos de gestão de permissões a serem desenvolvidos negam, por defeito, todo o acesso, necessitando de concessões explícitas para que utilizadores e papéis (*roles*) acedam a cada recurso.

A9 Proteção de Camada de Transporte Insuficiente – As aplicações frequentemente não autenticam, encriptam ou protegem a confidencialidade e integridade de tráfego de rede sensível. Quando o fazem, algumas vezes empregam algoritmos fracos, utilizam certificados expirados ou inválidos, ou não os utilizam corretamente.

Prevê-se que no contexto do *Personal Health Record* todo o tráfego seja realizado por SSL (tanto de *frontend* como *backend*) porque todas as páginas e dados são privados. De notar ainda que o ODBC/JDBC *standard* também vai ser configurado para usar SSL já que por defeito os dados são comunicados de forma não criptografada. Apesar do SSL prejudicar o desempenho, e por tal alguns *websites* apenas o utilizarem em páginas críticas, o OWASP considera que é vantajoso requerer SSL em todo a aplicação. Desta forma, evitam-se problemas de desenho da aplicação e riscos de segurança, já que não utilizar SSL em toda a aplicação torna mais fácil que ocorram falhas que expõem o id de sessão e outros dados sensíveis.

A10 Redireccionamentos e *Forwards* Não Validados – Sem validação adequada um ataque pode redirecionar a vítima para *websites* de *phishing* ou *malware* (por exemplo se a página de destino do *redirect* for determinada por um parâmetro não validado que seja alterado pelo ataque). Por outro lado, podem ser utilizados *forwards* para aceder a páginas não autorizadas.

Antes de mais, sempre que possível, deve ser evitar o uso de *forward* e *redirect*. O *forward* foi possível evitar por completo já que nem sequer é suportado pela Play. De facto, na Play um pedido HTTP só pode invocar uma ação. Se for necessário invocar outra ação é necessário redirecionar o *browser* para o URL capaz de invocar essa ação. Desta forma o URL do *browser* é sempre consistente com a ação executada e a gestão do **Back/Forward/Refresh** é facilitada. O *redirect*, quando empregue, verifica a autorização e não utiliza parâmetros do utilizador na determinação do destino (ou pelo menos não usa parâmetros que não possam ser validados).

5.9.2 Questões Adicionais de Segurança Consideradas Importantes

***Logging* –** Tal como foi instruído pela direção, não houve ainda grande investimento nesta questão uma vez que o *logging* deve ser realizado de forma consistente em toda a plataforma para que possa haver monitorização, alertas e relatórios adequados. Adicionalmente, está planeada a criação de um módulo, ainda não desenhado, que irá agregar a informação de *logging* de toda a plataforma para que o seu estado possa ser monitorizado num ponto centralizado.

Uma vez definido o que deve ser registado, essa informação pode ser utilizada para identificar riscos de segurança e responsabilizar entidades pelos seus atos. Atualmente o sistema de *logging* da Generic Entity utiliza os mecanismos da Play para registar operações mal sucedidas, erros da aplicação, falhas e sucessos de autenticação, falhas de autorização, falhas de validação de *input*, uso de parâmetros inválidos, entre outras ocorrências que são tipicamente relevantes tanto do ponto de vista de teste como de monitorização em ambiente de produção. Salienta-se ainda que a Generic Entity tem sistema de controlo de versões de entidades que pode ser utilizado como instrumento de *logging* das alterações efetuadas.

Certificados de Cliente – Os certificados são documentos eletrónicos geralmente

utilizados para identificar um servidor ou instituição, mas que também podem ser utilizados para identificar um indivíduo. Num contexto de saúde a segurança é crítica e seria ideal poder contar com SSL mútuo que providencia os mesmos benefícios que SSL, com a adição de autenticação e não-repúdio do lado do cliente, através do uso de certificados digitais. Desta forma, por exemplo, um utente que quisesse dar permissões sobre os seus dados a um profissional de saúde podia ter mais garantias sobre a identidade do indivíduo. Contudo, devido a problemas de complexidade, custo e logística a maioria das aplicações *web* são desenhadas de forma a não requerer o uso de certificados do lado do cliente.

Dado que este é um problema prático muito conhecido foram criados métodos alternativos ao certificado digital do lado do cliente. Um desses métodos envolve a emissão de um cartão físico com uma grelha de valores aleatórios, em que um desses valores é pedido pelo servidor antes do utilizador poder efetuar determinadas operações (e já depois de estar devidamente autenticado). Este método tipicamente bloqueia todos os privilégios da conta se o utilizador falhar no valor pedido mais do que um determinado número de vezes. Outro método mais recente que começa a ganhar popularidade é o envio de uma mensagem com um código para o telemóvel do utilizador, que lhe permite efetuar uma operação e que apenas é valido durante um curto período de tempo.

Neste momento não é possível saber se o projeto irá utilizar certificados do lado do cliente, mas é interessante constatar que existem estratégias com menos custos e maior usabilidade para adicionar segurança à interação com a plataforma.

Não-Repúdio – A partir do momento em que o servidor tenha um certificado digital válido (para confirmar a sua identidade), cada utilizador tenha a sua própria conta e respetivas credenciais (e preferencialmente tenha ele próprio uma forma de certificação perante o servidor), os *cookies* sejam assinados com uma chave secreta, cada aplicação que aceda à API tenha a sua própria "Consumer Key" e "Consumer Secret", toda a informação seja trocada por SSL, em todas as operações seja verificada a autenticação e autorização e todas as ações dos utilizadores e aplicações sejam registadas, estará garantido, em princípio, que se pode identificar com não-repúdio a entidade que praticou cada operação.

5.10 Conclusão

O desenho da arquitetura exigiu preparação prévia, análise crítica e a tomada de decisões nada triviais. Felizmente, a fase de investigação inicial facilitou a descoberta, aprendizagem e deliberação das respostas mais corretas a alguns requisitos do projeto.

Os aspetos atípicos deste projeto levaram a um afastar dos padrões e estratégias mais comuns implementados nos sistemas *web* atuais e a uma consequente perda de vários benefícios e facilidades que lhes estão associados. Contudo, estes compromissos não foram em vão permitindo a construção de uma infraestrutura de *software* que suporta os requisitos do projeto e que é extremamente flexível, como já foi possível verificar.

Capítulo 6
Testes

Os testes de *software* permitem obter informação sobre a qualidade do sistema alvo. Criar conjuntos de teste automáticos para a aplicação é uma boa forma de a tornar robusta e permite desenvolver e fazer alterações com confiança e de forma ágil. Durante o processo de construção foram programados testes unitários, funcionais, de interface e de integração automáticos.

Nesta secção são ainda apresentados testes de desempenho que visaram analisar, por comparação ao ORM tradicional, se a solução criada é aplicável a um contexto real, ou se, porventura, o peso da geração de código seria inaceitável e incontornável.

6.1 Testes Unitários, Funcionais e de Integração

A Play já apresenta recursos para programação de testes unitários e funcionais, utilizando JUnit, e de interface *web* utilizando Selenium[22]. Foi ainda utilizado o módulo Cobertura [80], que é uma ferramenta Java que permite analisar e distinguir visualmente o código acedido do não acedido pelos testes. Este módulo foi utilizado durante o desenvolvimento para identificar partes da aplicação que deviam estar melhor testadas.

A Figura 17 apresenta a interface de execução dos testes com a Play.

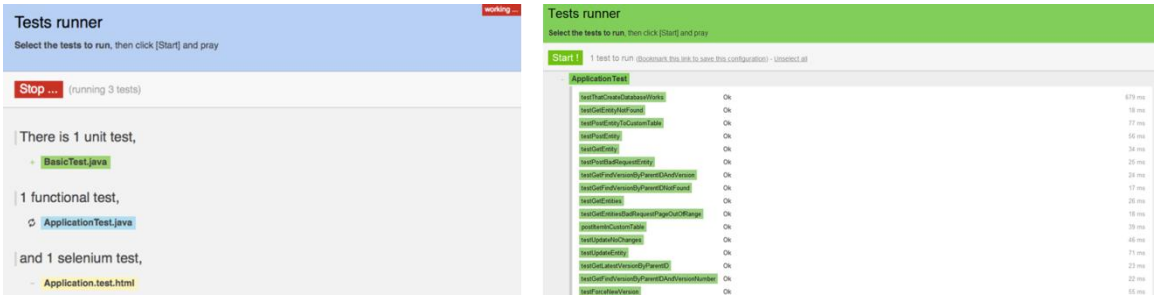

Figura 17 Exemplo da execução de testes com a framework Play.

A Figura 18 apresenta um exemplo da execução de testes através da interface de Selenium da Play na qual se pode visualizar e controlar a execução dos testes.

[22] Ferramenta de automatização de tarefas em *browsers*.

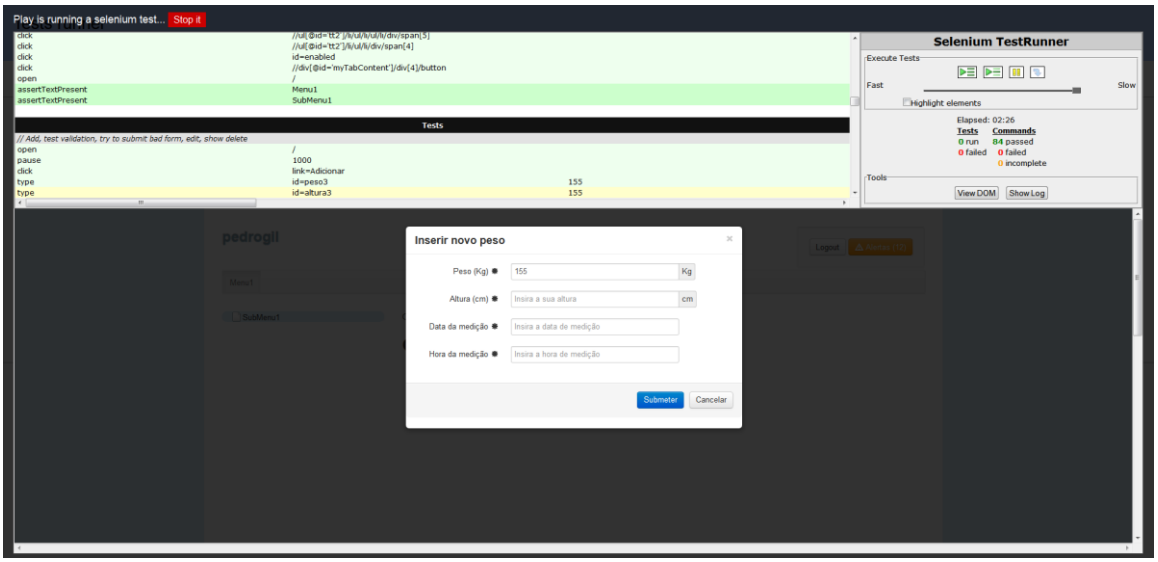

Figura 18 Exemplo de utilização da interface do Selenium na Play.

A especificação dos testes de sistema realizados encontra-se no documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)" na secção D1. Especificação de Testes de Sistema. Nesta secção os testes realizados são agrupados em 44 conjuntos, cada um podendo estar ainda subdividido em grupos de testes unitários, de interface, funcionais, de integração e de segurança.

Foram realizadas sessões de demonstração perante os membros da equipa técnica e de gestão de projeto no final de cada *sprint*. Para além disso ocorreram diversas demonstrações perante parceiros e consultores do plano de negócios que procuravam saber mais sobre os projetos. Nestas apresentações o uso da ferramenta foi feito ao vivo, sem auxílio de vídeos.

6.2 Testes de Desempenho e Utilização de Recursos

Para os testes de desempenho foi utilizado o Apache JMeter [81], que pode ser empregue para testar o desempenho de recursos estáticos e dinâmicos. O JMeter é um *software open-source* programado em Java que funciona como uma aplicação de *desktop*.

Dado que a Generic Entity é altamente configurável e flexível era esperado que o seu desempenho fosse significativamente inferior a soluções estáticas. Desta forma, era importante analisar se a sobrecarga desta solução poderia ser relativizada e ter um impacto aceitável face ao contexto geral de uma aplicação *web*.

O objetivo dos testes realizados foi comparar o desempenho da solução Generic Entity com uma solução de ORM tradicional (aqui foi utilizada a Play porque a Generic Entity é construída com recurso à Play, o que torna as condições de comparação mais parecidas). Considera-se desnecessário comparar ainda com uma abordagem que não abstrai do acesso à base de dados, isto é, simplesmente construir as *queries* manualmente, uma vez que hoje em dia raramente tal solução é considerada ainda que, potencialmente, evite o peso dos mecanismos de abstração.

6.2.1 Configurações dos Testes

Os testes foram executados com as seguintes configurações:

- Com e sem uso de mecanismos de *cache*.
- Com base de dados vazia e com base de dados populada com 100.000 registos criados de forma aleatória, mas igual para ambas as aplicações a serem comparadas.
- Com uma entidade de 5 atributos (3 *strings*, 1 inteiro e 1 *text*) e outra com 15 (8 *strings*, 2 inteiros, 1 *text*, 1 data, 1 *float*, 1 *long* e 1 tempo).

Optou-se por não utilizar pedidos concorrentes (múltiplos utilizadores) porque nos primeiros testes se verificou que estas condições causavam desvios entre execuções iguais e não ajudavam na comparação de abordagens. Cada plano de teste foi corrido três vezes para garantir que não existiam grandes afastamentos nos resultados e cada resultado de teste era a média da execução da operação em causa 500 vezes.

Para representar a abordagem ORM foi criada uma aplicação em Play onde se define um modelo com os mesmos atributos de uma entidade guardada na Generic Entity.

O ambiente de testes e resultados podem ser consultados em detalhe no documento "TICE-GenericEntity Testes de Desempenho".

6.2.2 Análise Geral dos Testes de Comparação

Os testes a comparar o espaço ocupado pelas duas soluções revelou que a diferença é negligenciável. As colunas com valores a *null* não afetam significativamente o espaço necessário pela Generic Entity utilizando o SGBD PostgreSQL. Nos testes realizados a maior diferença de espaço observada foi de 1MB.

Os resultados seguintes apresentam resumidamente os tempos médios de execução dos testes de comparação de desempenho (cada resultado é a média de 500 ciclos de teste). Para resultados mais detalhados com valores de desvio padrão, tempo máximo, tempo mínimo, entre outros, consulte o documento anexo "TICE-GenericEntity Testes de Desempenho".

Para os testes de inserção de 50 registos os resultados obtidos foram surpreendentes e melhores do que o esperado (Gráfico 1). De facto, acreditava-se que para esta operação a Generic Entity fosse significativamente mais lenta uma vez que tem que construir iterativamente o objeto a inserir na base de dados verificando cada parâmetro recebido no pedido HTTP face aos atributos definidos na configuração, e o mesmo para a construção da *query* de inserção que sendo gerada a partir de uma configuração dinâmica seria de esperar que tivesse uma sobrecarga.

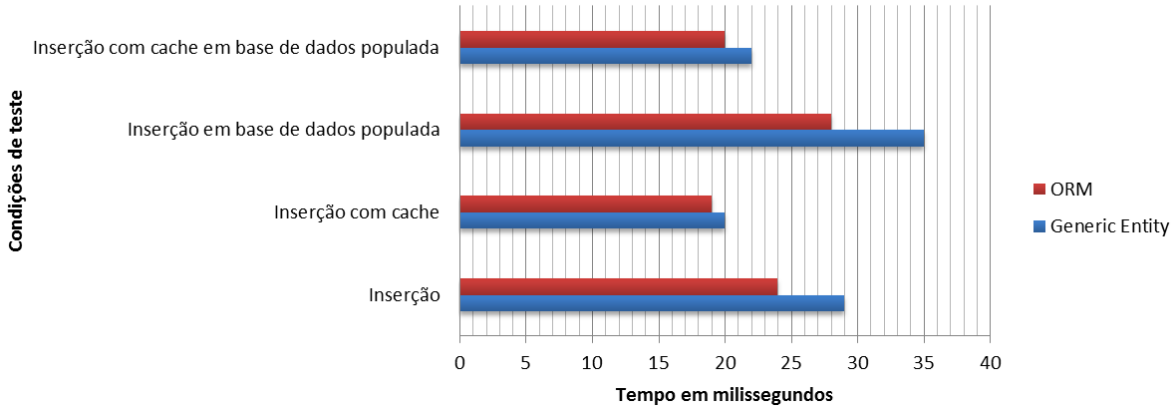

Gráfico 1 Comparação do tempo médio de inserção de 50 registos.

De acordo com os resultados estas operações extra são, aparentemente, muito rápidas. Adicionalmente, ao testar utilizando uma entidade estendida (com 15 atributos em vez de 5) verificou-se que o impacto da extensão era insignificante (como se pode observar no Gráfico 2) logo a solução utilizada não está limitada em termos de desempenho pelo número de atributos da entidade.

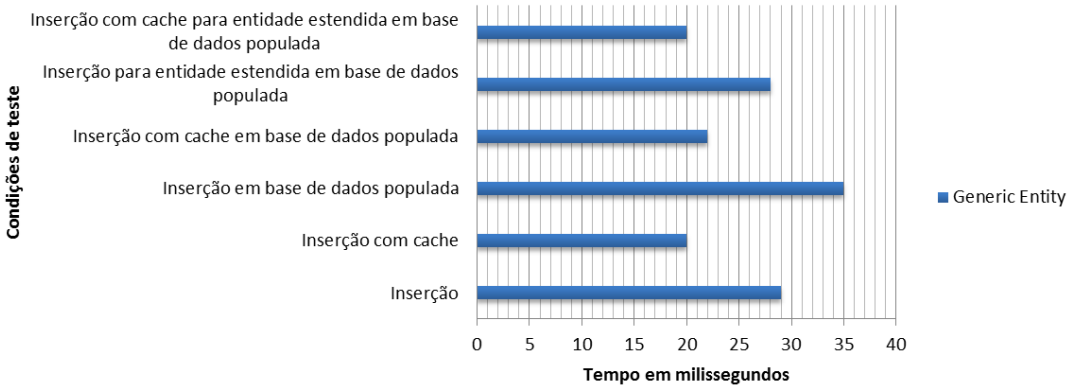

Gráfico 2 Comparação dos tempos de inserção da entidade inicial (com 5 atributos) e da entidade estendida (com 15 atributos) na Generic Entity.

O teste de obtenção de registos revelou que a Generic Entity era um pouco mais lenta ao correr este teste com a base de dados populada (Gráfico 3). Apesar de não ser uma diferença significativa, menos de 10 milissegundos de diferença para obtenção de 50 registos, suscitou curiosidade pelo que foram analisadas as *queries* em causa, que revelaram que a Play não vai buscar tantos atributos (não tem atributos de configuração como data de criação e atualização) e não ordena os resultados, enquanto a Generic Entity por defeito ordena os resultados pelo *timestamp* de criação, já que devolve sempre os registos de forma paginada.

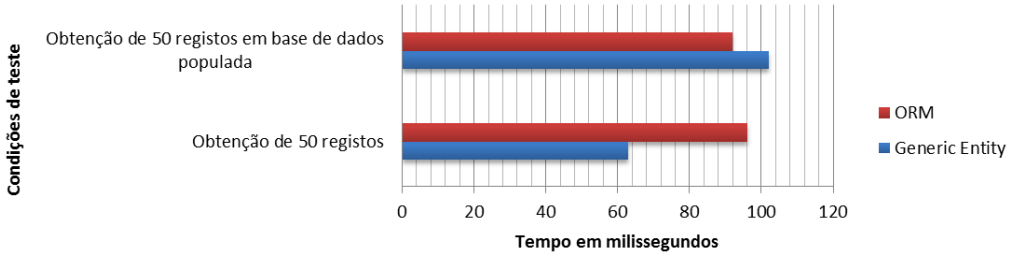

Gráfico 3 Comparação do tempo médio de obtenção de 50 registos.

Os testes de tempo de criação de um formulário de inserção pretenderam dar a conhecer o tempo de criação de um formulário com a Generic Entity em comparação com o tempo que a Play demora a devolver esse mesmo formulário já criado. Isto é, a Generic Entity interpretou a configuração de uma vista de inserção para gerar o HTML e JavaScript de validação de um formulário e a aplicação de comparação simplesmente devolveu esse formulário (pré-criado) como o faria com qualquer página estática.

Nestes testes a criação de um formulário de inserção utilizando a Generic Entity é sensivelmente 10 vezes mais lenta que a apresentação de um formulário estático (passa de 1 milissegundo para 10 milissegundos no caso de teste) como se pode observar no Gráfico 4. Mais uma vez este resultado foi surpreendentemente positivo dada a natureza da comparação. De facto, era já sabido que a Generic Entity seria mais lenta e é perfeitamente aceitável que seja 10 vezes mais lenta já que existem mecanismos para mitigar esta diferença. Com efeito, utilizando *cache* na Generic Entity e na aplicação de comparação, os resultados são, naturalmente, iguais.

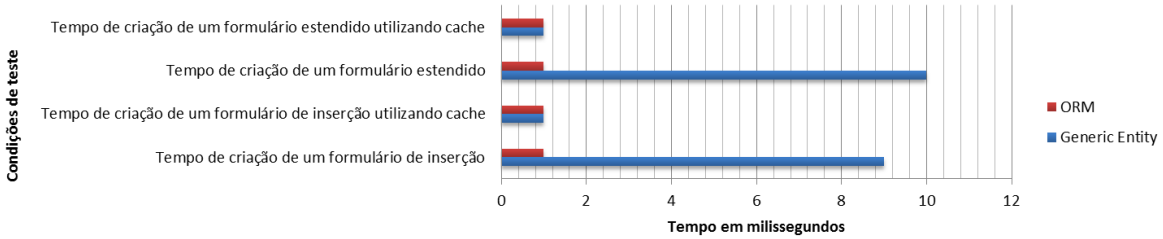

Gráfico 4 Comparação dos tempos de criação de um formulário de inserção.

Nestes testes a *cache* não é apenas utilizada para guardar o resultado do *unmarshall*[23] da Generic Entity. A Play possibilita o *cache* de partes do *template*, algo que também foi empregue nestes testes. Este mecanismo é extremamente útil neste contexto já que os formulários tipicamente não são algo altamente personalizado, mas sim partilhados por vários utilizadores, pelo que o sistema de *cache* é muitíssimo eficaz. A mesma conclusão é válida para uma vista de listagem de dados que, ao ser gerada no *template*, pode ser guardada em *cache* para todo a aplicação. Esta afirmação pode parecer estranha, uma vez que a listagem contém registos específicos que dependem, por exemplo, do utilizador. Contudo, as listagens/tabelas na Generic Entity são criadas sem registos; estes só são carregados através de um pedido AJAX posterior à criação da página pelo servidor.

6.3 Conclusão

Como se pôde verificar apesar do seu dinamismo e flexibilidade a Generic Entity não apresenta uma grande sobrecarga face a soluções tradicionais como o ORM. Naturalmente a geração de um formulário é mais lenta que a simples apresentação de um formulário estático, mas é um peso relativamente insignificante em condições que contemplem a latência da rede. De qualquer das formas, foi comprovado que o uso de um sistema de *cache* para mitigar a sobrecarga de geração de vistas é extremamente eficaz. Na maioria dos casos é possível fazer *cache* das operações mais pesadas da Generic Entity, nomeadamente:

- O pedido da configuração de uma entidade ao Core (que é igual para todos os utilizadores).
- O pedido da entidade feito pelo Core ao sistema de gestão de bases de dados.
- O *marshall* e *unmarshal* dessa configuração do lado do Core e Viewer.
- É possível fazer *caching* a partes dos *templates* da aplicação, logo a própria geração de formulários ou vistas sobre os dados pode ser toda guardada em *cache* para pedidos seguintes, nos quais em alguns casos serve para todos os utilizadores e noutros o *caching* é específico de cada utilizador (conteúdo processado especificamente para um utilizador), empregando-se o seu id ou o id da sessão para compor a chave de acesso à *cache*.

Contudo, este aumento de desempenho consegue-se sempre à custa da redução da atualidade dos dados [73]. No caso desta aplicação uma consequência imediata do uso de *cache* para, por exemplo, guardar a configuração das entidades, é a perda da atualização imediata da interface quando as configurações são alteradas. De facto, se o sistema de *cache* tiver a configuração de uma entidade a ser guardada por 15 minutos só ao fim desse tempo é que seriam refletidas as mudanças de configuração. Esta condição é aceitável para a maioria dos casos, porém não é prática durante o desenvolvimento da aplicação.

[23] *Marshall* e *unmarshall* referem-se respetivamente a transformar a representação em memória de um objeto para um formato de dados como JSON e vice-versa.

Capítulo 7
Resultados

Uma modelação típica das restrições de um projeto de *software* é conhecida como o **Triângulo de Gestão de Projetos** (Figura 19) e apresenta como lados do triângulo as variáveis: **custo**, **âmbito** e **tempo**. Tipicamente um lado deste triângulo não pode ser alterado sem afetar os restantes. Um refinamento deste modelo separa ainda o **âmbito/alcance** da **qualidade** do projeto.

Figura 19 Triângulo de Gestão de Projetos.

A abordagem da Generic Entity permite influenciar positivamente atributos de desenvolvimento que normalmente entram em conflito – o **tempo** de desenvolvimento é reduzido, a **qualidade** da aplicação aumenta, tudo isto sem consumo de recursos (o que diminui o **custo**). Desta forma, o **alcance** do projeto pode também tornar-se mais ambicioso. As razões de poupança no tempo de desenvolvimento e aumento de qualidade derivadas do uso da Generic Entity são resumidas abaixo:

- O código é apenas escrito uma vez e cada pedaço de informação só tem uma representação no sistema.
- Os eventuais *bugs* tornam-se mais visíveis (replicam-se cada vez que o código é reutilizado), o que diminui o tempo de desenvolvimento e facilita a garantia de qualidade.
- Para correção de erros ou adição de novas funcionalidades apenas é necessário fazer a alteração num local.
- O código é mais robusto e o comportamento é mais coerente, já que é mais difícil esquecer algum tipo de validação ou mecanismo de segurança (como por exemplo a inclusão de um *token* de autenticidade num formulário para evitar ataques *Cross-Site Request Forgery*).
- É possível adaptar a mudanças nas entidades de negócio com mínima latência e máxima flexibilidade.

7.1 Trabalho Realizado e Fronteira do Projeto

O alcance do projeto ficou claro e foi cumprido – exatamente o que tinha que ser concluído no estágio e o que devia ficar de fora. Os contornos do projeto também ficaram muito bem definidos: o estagiário desenvolveu autonomamente todos os aspetos que se prendem com a Generic Entity, as tarefas de integração foram concretizadas em colaboração e envolveram a definição de novos requisitos, troca de *feedback* e planeamento por parte dos indivíduos envolvidos, contudo a implementação propriamente dita dos mecanismos dentro de cada um dos projetos foi concretizada pelos responsáveis de cada sistema. O *design* e *layout* atuais também foram da responsabilidade do estagiário, mas serão refeitos quando forem recebidas as indicações e elementos da empresa de *web design* que foi contratada para os TICE, a

bürocratik. Todavia, a interface *web* da aplicação desenvolvida pode ser consultada no documento "TICE-GenericEntity – gestor de entidades e gerador de formulários (Anexos)", no Apêndice E Interface da Aplicação.

Os resultados atuais do trabalho realizado neste estágio são:

- O problema foi analisado, compreendido e descrito.
- Foi realizada uma análise do estado da arte.
- Foram listados dos requisitos de sistema de acordo com os objetivos do projeto.
- As tecnologias principais a utilizar foram escolhidas, e a escolha foi fundamentada.
- Foi concebida e descrita a arquitetura geral do sistema.
- Foi concebida a forma de funcionamento interna da Generic Entity.
- A solução concebida foi implementada.
- Foram realizados testes à robustez e desempenho do sistema.
- O sistema foi integrado com outros projetos, como o Sensor Care e o portal/Appbase.
- O sistema foi demonstrado no final de cada *sprint* e houve uma apreciação positiva do mesmo por parte da gerência.

Por fim, é também importante destacar que foi submetido e aceite para a conferência *9th International Workshop on Wearable, Micro and Nano Technologies for the Personalized Health, pHealth 2012* [82], um artigo para demonstração do projeto (que se encontra em anexo). Dada a natureza do evento, esta demonstração focou os benefícios da Generic Entity especificamente para a área de registos clínicos eletrónicos. A apresentação do projeto ocorreu no dia 27 de Junho de 2012, no Porto, onde foram demonstrados os mecanismos e dinamismo da solução criada já integrada como aplicação do portal. A Figura 20 apresenta a notícia sobre o evento publicada no *website* do Laboratório de Automática e Sistema do IPN [83].

Figura 20 Notícia publicada no website do Laboratório de Automática e Sistema do Instituto Pedro Nunes.

Capítulo 8
Conclusões

O principal objetivo deste estágio foi o desenho e implementação de um sistema que permite gerir entidades e gerar os componentes para visualizar e manipular os registos dessas entidades de forma dinâmica mesmo em ambiente de produção. De forma complementar foram concebidos e implementados mecanismos para criar de raiz, e modificar dinamicamente, aplicações de visualização de dados completas com menus/submenus que contêm componentes de diversos tipos.

8.1 Desafios e Contributos

> *"When woodworkers are faced with the task of producing the same thing over and over, they cheat. They build themselves a jig or a template. If they get the jig right once, they can reproduce a piece of work time after time. The jig takes away complexity and reduces the chances of making mistakes, leaving the craftsman free to concentrate on quality.*
>
> *As programmers, we often find ourselves in a similar position. We need to achieve the same functionality, but in different contexts. We need to repeat information in different places. Sometimes we just need to protect ourselves from carpal tunnel syndrome by cutting down on repetitive typing*
>
> *In the same way a woodworker invests the time in a jig, a programmer can build a code generator. Once built, it can be used throughout the life of the project at virtually no cost.*
>
> ***Tip 29** Write Code That Writes Code"*
>
> *The Pragmatic Programmer: From Journeyman to Master, Andy Hunt e Dave Thomas* [72]

Neste projeto foi criado um gerador de interfaces embutido num ambiente dinâmico que reage de acordo com a configuração tanto na forma como armazena os dados como na forma como os disponibiliza. Criar este componente revelou-se um enorme desafio de engenharia, sendo necessário compreender as necessidades e adequar o conceito a um sistema real, que pode ainda ser reaproveitado em diversos outros cenários.

Um conceito importante que justifica a criação deste gerador de código é o *Money for Nothing*. Existem diversas características e componentes comuns em sistemas de informação e tarefas repetitivas envolvidas na sua implementação. Um investimento na criação de ferramentas que automatizem e facilitem ao máximo estas tarefas irá levar a que, eventualmente, os custos de implementação de novos projetos sejam mínimos sendo o lucro maximizado (daí o princípio de ser pago por trabalho sem custos – *Money for Nothing*). Nos exemplos mencionados o conceito de *Money for Nothing* seria aplicado a uma parte dos projetos que foi criada de raiz e podia ter sido automatizado, sendo que as características mais específicas e adicionais de cada projeto podiam ser programadas normalmente dentro do sistema Generic Entity. O exemplo mais acessível desta questão está no próprio painel de administração do Viewer da Generic Entity, onde foi conseguida uma poupança de tempo de implementação ao utilizar a Generic Entity para geração de vistas e formulários, por exemplo para a gestão de conteúdo da aplicação de visualização. Por outras palavras, a Generic Entity foi usada como suporte para implementação do seu próprio painel de administração.

8.2 Experiência Global

Este estágio revelou-se uma importante experiência pessoal e profissional. Integrar uma

equipa de desenvolvimento motivada, experiente, profissional e exigente colocou à prova e desenvolveu tanto conhecimentos técnicos, como capacidades interpessoais valiosas adquiridas ao longo do curso de Engenharia Informática.

A nível técnico o projeto exigiu a resposta a requisitos desafiantes e interessantes que obrigaram a explorar e tomar percursos menos convencionais. Os recursos a analisar no estado da arte obrigaram a uma investigação elaborada, uma vez que muitas estratégias interessantes não estão completamente documentadas ou acessíveis, por estarem implementadas em produtos comerciais.

A possibilidade de estudar o universo das bases de dados NoSQL foi extremamente interessante, mas, em algumas fases, igualmente esmagadora devido ao enorme número de produtos (atualmente mais de uma centena), à variedade de soluções e abordagens, à inexistência de uma taxonomia oficial e unificadora das diferentes visões e à rápida evolução que torna diariamente obsoleta e contraditória informação relativamente recente. Ainda durante o estágio tive oportunidade de integrar e testar o MongoDB na Play, bem como começar o desenho da solução Generic Entity no contexto desta base de dados de documentos. Gostaria de continuar a explorar esta vertente para enriquecer o meu conhecimento e poder ter uma visão mais clara das suas vantagens, mas reconheço que, atualmente para este projeto, tal teria poucas vantagens de negócio.

A nível não técnico houve a oportunidade de aplicar na prática conceitos de processos de engenharia aprendidos no curso, num projeto grande e num contexto ambicioso. A análise cuidada do estado da arte e dos requisitos, e o desenho informado da arquitetura, levaram a uma enorme poupança de tempo e uma correta resposta às necessidades do projeto.

8.3 Principais Obstáculos

Ao longo do primeiro semestre surgiram diversos obstáculos ao estágio. De uma perspetiva de planeamento o maior contratempo foi a redefinição iterativa dos objetivos do estágio durante uma longa fase. A falta de informação atempada por parte de alguns parceiros causou um enorme atraso.

O segundo semestre apresentou muitas obrigações e pouco tempo para colmatar a falta de experiência com diversas tecnologias que se revelaram necessárias. Apesar de por vezes estas situações terem obrigado ao refazer de partes do trabalho, à medida que se adquiriam novos conhecimentos sobre as tecnologias e conceitos envolvidos, estes desafios foram os mais interessantes e proporcionaram excelentes oportunidades de aprendizagem.

Numa perspetiva global faz-se notar a complexidade organizacional do projeto derivada principalmente da existência de 24 parceiros no consórcio. Também a descentralização de poder no projeto, a alto nível pela existência de vários parceiros, e a baixo nível pela presença na equipa de diferentes gestores com visões algumas vezes diferentes da mesma questão, levou a alguma ineficiência na comunicação e tomada de decisões. Contudo considera-se que o conflito local de opiniões levou sempre, eventualmente, a uma ponderação mais profunda sobre as questões em causa.

8.4 Trabalho Futuro

Antes do fim do estágio os responsáveis do subprojecto Mind.Care começaram a estudar a API da Generic Entity para integrar o seu trabalho com a plataforma. Parte deste

subprojecto consistirá numa aplicação para iPhone que irá recolher dados de sensores (especificamente direcionados para utentes com doenças crónicas, como sensores que detetam quedas) que serão armazenados e trabalhados na plataforma. Este cenário é muito semelhante ao do Sensor Care e, tanto quanto foi analisado, numa fase inicial apenas será necessário criar novas entidades para dar suporte a este projeto.

Um próximo passo importante é integrar o servidor de OAuth com a Generic Entity, da mesma forma que foi feito no portal, e começar a definir e integrar o sistema de permissões e os mecanismos adicionais de segurança, em parceria com a HIS que é oficialmente responsável por estas tarefas. Em princípio será ainda esta empresa que se encarregará da especificação e configuração da primeira versão oficial do PHR.

Apesar de funcional, o editor desenvolvido para a Generic Entity não tem ainda a mesma capacidade de configuração que se consegue com a aplicação desenvolvida para gerar as configurações de entidades programaticamente instanciando as classes de configuração e gerando a respetiva representação em JSON. Esta situação está exatamente de acordo com o esperado uma vez que só se pretendia o desenvolvimento de uma primeira versão do editor. Dado que este provou ser uma forma mais rápida de editar e criar entidades, e tem uma barreira técnica muito inferior, vai haver um investimento para implementar as capacidades de configuração que faltam.

Deverá estar para breve começar a aplicar esforços para melhorar a aparência e usabilidade da Generic Entity (especificamente do PHR) através do trabalho que está a ser realizado pela empresa bürocratik.

Referências

[1] TICE, "Pólo de Competitividade das Tecnologias de Informação Comunicação e Eletrónica TICE.PT." [Online]. Available: tice.pt. [Accessed: 19-Dec-2011].

[2] L. Xie, C. Yu, L. Liu, and Z. Yao, "XML-based Personal Health Record system," in *2010 3rd International Conference on Biomedical Engineering and Informatics*, 2010, pp. 2536-2540.

[3] "scrum.org," 2011. [Online]. Available: http://www.scrum.org/. [Accessed: 21-Nov-2011].

[4] S. Chacon, "git - the fast version control system." [Online]. Available: http://git-scm.com/. [Accessed: 21-Nov-2011].

[5] V. Driessen, "gitflow," 2011. [Online]. Available: https://github.com/nvie/gitflow. [Accessed: 21-Nov-2011].

[6] V. Driessen, "A successful Git branching model," 2010. [Online]. Available: http://nvie.com/posts/a-successful-git-branching-model/. [Accessed: 21-Nov-2011].

[7] J. Kreeftmeijer, "Why aren't you using git-flow?," 2010. [Online]. Available: http://jeffkreeftmeijer.com/2010/why-arent-you-using-git-flow/. [Accessed: 21-Nov-2011].

[8] "Personal Health Record and Personal Health Record System, A Report Recommendation from the National Committee on Vital and Health Statistics," 2006.

[9] P. C. Tang, J. S. Ash, D. W. Bates, J. M. Overhage, and D. Z. Sands, "Personal health records: definitions, benefits, and strategies for overcoming barriers to adoption.," *Journal of the American Medical Informatics Association : JAMIA*, vol. 13, no. 2, pp. 121-6.

[10] K. D. Mandl and I. S. Kohane, "Tectonic shifts in the health information economy.," *The New England Journal of Medicine*, vol. 358, no. 16, pp. 1732-7, Apr. 2008.

[11] A. Sunyaev, D. Chornyi, C. Mauro, and H. Krcmar, "Evaluation Framework for Personal Health Records: Microsoft HealthVault Vs. Google Health," in *2010 43rd Hawaii International Conference on System Sciences*, 2010, pp. 1-10.

[12] "Meu Sapo Saúde." [Online]. Available: https://meu.saude.sapo.pt/. [Accessed: 21-Oct-2011].

[13] "About Google Health." [Online]. Available: http://www.google.com/intl/pt-PT/health/about/. [Accessed: 21-Oct-2011].

[14] "Google health Frequently Asked Questions." [Online]. Available: http://www.google.com/intl/en-US/health/faq.html. [Accessed: 28-Oct-2011].

[15] "Continuity of Care Record (CCR)." [Online]. Available: http://www.ccrstandard.com. [Accessed: 28-Oct-2011].

[16] D. Blankenhorn and ZDNET, "Microsoft HealthVault is nothing like Google Health," 2008. [Online]. Available: http://www.zdnet.com/blog/healthcare/microsoft-healthvault-is-nothing-like-google-health/742. [Accessed: 01-Oct-2012].

[17] M. Cross, "Goliath moves into healthcare records," *BMJ Clinical Research Ed.*, vol. 335, 2007.

[18] B. Dolan and Mobihealthnews, "10 Reasons why Google Health failed," 2011. [Online]. Available: http://mobihealthnews.com/11480/10-reasons-why-google-health-failed/. [Accessed: 12-Jan-2012].

[19] Z. Erdos and CloudAve, "The Sorry State of Health 2.0 – Google Health & Microsoft HealthVault," 2009. [Online]. Available: http://www.cloudave.com/2452/the-sorry-state-of-health-2-0-google-health-microsoft-healthvault/. [Accessed: 12-Jan-2012].

[20] "Microsoft Sharepoint 2010," 2010. [Online]. Available: http://sharepoint.microsoft.com/pt-pt/Pages/default.aspx. [Accessed: 24-Nov-2011].

[21] Mmuro, "WordPress › Visual Form Builder « WordPress Plugins," 2011. [Online]. Available: http://wordpress.org/extend/plugins/visual-form-builder/. [Accessed: 09-Dec-2011].

[22] TruthMedia and J. Warkentin, "WordPress › FormBuilder « WordPress Plugins," 2011. [Online]. Available: http://wordpress.org/extend/plugins/formbuilder/. [Accessed: 09-Dec-2011].

[23] TruthMedia and J. Warkentin, "TruthMedia » FormBuilder: Experts in Online Ministry," 2011. [Online]. Available: http://truthmedia.com/wordpress/formbuilder/. [Accessed: 09-Dec-2011].

[24] J. Gilchrist, M. Frize, C. M. Ennett, and E. Bariciak, "Performance Evaluation of Various Storage Formats for Clinical Data Repositories," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 10, pp. 3244-3252, Oct. 2011.

[25] J. Krause, C. Langhirt, A. Sterff, B. Pehlke, and M. Doring, *SharePoint 2010 as a Development Platform*. Apress; 1 edition, 2010, p. 800.

[26] HL7, "Health Level Seven International." [Online]. Available: http://www.hl7.org/. [Accessed: 30-Apr-2012].

[27] M. Yuksel and A. Dogac, "Interoperability of Medical Device Information and the Clinical Applications: An HL7 RMIM based on the ISO/IEEE 11073 DIM," *IEEE Transactions on Information Technology in Biomedicine*, vol. 15, no. 4, pp. 557-566, Jul. 2011.

[28] S. Gaion, S. Mininel, F. Vatta, and W. Ukovich, "Design of a domain model for

clinical engineering within the HL7 Reference Information Model," in *2010 IEEE Workshop on Health Care Management (WHCM)*, 2010, pp. 1-6.

[29] Ringholm, "HL7 v3 RIM based applications: an unintended side effect." [Online]. Available: http://www.ringholm.de/column/hl7_rim_based_applications_an_unintended_side_effect.htm. [Accessed: 30-Apr-2012].

[30] J. Gilchrist, M. Frize, C. M. Ennett, and E. Bariciak, "Performance Evaluation of Various Storage Formats for Clinical Data Repositories," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 10, pp. 3244-3252, Oct. 2011.

[31] Mirth Corporation, "mirth - Powering Healthcare Interoperability." [Online]. Available: http://www.mirthcorp.com/. [Accessed: 30-Apr-2012].

[32] M. Alam, M. Hussain, M. Afzal, M. Maqbool, H. F. Ahmad, and S. Razzaq, "Design and Implementation of HL7 V3 Standard-Based Service Aware System," in *2011 Tenth International Symposium on Autonomous Decentralized Systems*, 2011, pp. 420-425.

[33] W3C Ubiquitous Web domain, "Extensible Markup Language (XML)." [Online]. Available: http://www.w3.org/XML/. [Accessed: 19-Dec-2011].

[34] "Introducing JSON." [Online]. Available: http://json.org/. [Accessed: 19-Dec-2011].

[35] J. Wagner, "The Increasing Importance of APIs in Web Development | Nettuts+," 2011. [Online]. Available: http://net.tutsplus.com/articles/news/the-increasing-importance-of-apis-in-web-development/. [Accessed: 20-May-2012].

[36] A. Popescu, "myNoSQL - Getting Started with NoSQL," 2010. [Online]. Available: http://nosql.mypopescu.com/post/807203888/getting-started-with-nosql. [Accessed: 02-Dec-2011].

[37] G. Harrison, "TechRepublic - 10 things you should know about NoSQL databases," 2010. [Online]. Available: http://www.techrepublic.com/blog/10things/10-things-you-should-know-about-nosql-databases/1772. [Accessed: 03-Nov-2011].

[38] OmniTI, "Scalable Internet Architecture," 2010. [Online]. Available: http://www.slideshare.net/postwait/scalable-internet-architecture. [Accessed: 19-Dec-2011].

[39] Bundlr, "Bundlr - Bundle and share web content easily," 2011. [Online]. Available: http://gobundlr.com/. [Accessed: 03-Dec-2011].

[40] S. Santos, "sergiosantos.info." [Online]. Available: http://sergiosantos.info/tagged/mongodb. [Accessed: 03-Dec-2011].

[41] Boxed Ice and D. Mytton, "Boxed Ice Blog - Choosing a non-relational database; why we migrated from MySQL to MongoDB," 2009. [Online]. Available: http://blog.boxedice.com/2009/07/25/choosing-a-non-relational-database-why-we-migrated-from-mysql-to-mongodb/. [Accessed: 03-Dec-2011].

[42] Boxed Ice, "Boxed Ice Blog - MongoDB," 2011. [Online]. Available: http://blog.boxedice.com/mongodb/. [Accessed: 03-Dec-2011].

[43] I. Urban Airship, "Urban Airship powering modern mobile," 2011. [Online]. Available: http://urbanairship.com/. [Accessed: 03-Dec-2011].

[44] M. Schurter, "schmichael's blog - schmongodb slides from Update Portland," 2011. [Online]. Available: http://blog.schmichael.com/2011/02/02/schmongodb-slides-from-update-portland/. [Accessed: 03-Dec-2011].

[45] M. Schurter, "schmichael's blog - Failing with MongoDB," 2011. [Online]. Available: http://blog.schmichael.com/2011/11/05/failing-with-mongodb/. [Accessed: 03-Dec-2011].

[46] "foursquare," 2011. [Online]. Available: https://pt.foursquare.com/. [Accessed: 19-Dec-2011].

[47] N. Folkman, "So, that was a bummer. | Foursquare Blog," 2010. [Online]. Available: http://blog.foursquare.com/2010/10/05/so-that-was-a-bummer/.

[48] T. Hoff, "Troubles with Sharding - What can we learn from the Foursquare Incident?," 2010. [Online]. Available: http://highscalability.com/blog/2010/10/15/troubles-with-sharding-what-can-we-learn-from-the-foursquare.html. [Accessed: 19-Dec-2011].

[49] E. Gunderson, "Two Reasons You Shouldn't Use MongoDB." [Online]. Available: http://ethangunderson.com/blog/two-reasons-to-not-use-mongodb/. [Accessed: 03-Dec-2011].

[50] L. Montanez, "MongoDB 2.0 Should Have Been 1.0," 2011. [Online]. Available: http://luigimontanez.com/2011/mongodb-2.0-should-have-been-1.0/. [Accessed: 03-Dec-2011].

[51] Health Innovation Systems SA, "HIS health innovation systems," 2011. [Online]. Available: http://www.his.pt/. [Accessed: 03-Dec-2011].

[52] R. Olivieri, "Implement business logic with the Drools rules engine," 2008. [Online]. Available: http://www.ibm.com/developerworks/java/library/j-drools/. [Accessed: 28-Apr-2012].

[53] F. Marchioni, "JBoss Drools Tutorial." [Online]. Available: http://www.mastertheboss.com/jbpm/45-jboss-drools-1.html. [Accessed: 28-Apr-2012].

[54] GigaSpaces, "Drools Rule Engine Integration - Solutions and Best Practices - GigaSpaces Documentation Wiki." [Online]. Available: http://www.gigaspaces.com/wiki/display/SBP/Drools+Rule+Engine+Integration. [Accessed: 30-Apr-2012].

[55] I. Infoworld and S. Núñez, "ILOG JRules 6.5 brings rules to SOA," 2007. [Online]. Available: http://www.infoworld.com/d/developer-world/ilog-jrules-65-brings-

rules-soa-468. [Accessed: 02-Mar-2012].

[56] IBM, "WebSphere ILOG JRules BRMS - Flexible, governed decision automation." [Online]. Available: http://www-01.ibm.com/software/integration/business-rule-management/jrules-family/. [Accessed: 02-Mar-2012].

[57] I. Infoworld and S. Núñez, "First look: JBoss Drools grows up, and out," 2009. [Online]. Available: http://www.infoworld.com/d/developer-world/first-look-jboss-drools-grows-and-out-137?page=0,0. [Accessed: 02-Mar-2012].

[58] F. I. Corporation, "FICO® Blaze Advisor® business rules management." [Online]. Available: http://www.fico.com/en/Products/DMTools/Pages/FICO-Blaze-Advisor-System.aspx. [Accessed: 02-Mar-2012].

[59] Microsoft Corporation and J. Willis, "Introduction to the Windows Workflow Foundation Rules Engine," 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa480193.aspx. [Accessed: 02-Mar-2012].

[60] EsperTech Inc., "Esper - Complex Event Processing." [Online]. Available: http://esper.codehaus.org/. [Accessed: 02-Mar-2012].

[61] Apache Software Foundation, "Apache Jena." [Online]. Available: http://jena.apache.org/. [Accessed: 02-Mar-2012].

[62] P. Browne, "Give Your Business Logic a Framework with Drools - O'Reilly Media." [Online]. Available: http://onjava.com/pub/a/onjava/2005/08/03/drools.html?page=2. [Accessed: 29-Apr-2012].

[63] I. OpenRules, "OpenRules Business Decision Management System." [Online]. Available: http://openrules.com/. [Accessed: 02-Mar-2012].

[64] JBoss Community, "Drools - The Business Logic integration Platform." [Online]. Available: http://www.jboss.org/drools. [Accessed: 02-Mar-2012].

[65] Stack Overflow, "What are some real world examples of using Drools?," 2010. [Online]. Available: http://stackoverflow.com/questions/3453111/what-are-some-real-world-examples-of-using-drools. [Accessed: 28-Apr-2012].

[66] "TICE.Healthy Candidatura - SISTEMA DE INCENTIVOS À INVESTIGAÇÃO E DESENVOLVIMENTO TECNOLÓGICO - PROJECTOS DE I&DT EMPRESAS MOBILIZADORES," 2009.

[67] M. Machado and J. Quintas, "Arquitetura geral da plataforma We.Can versão 0.3," 2011.

[68] zenexity, "Asynchronous programming with HTTP." [Online]. Available: http://www.playframework.org/documentation/1.2.4/asynchronous. [Accessed: 23-May-2012].

[69] Sun Microsystems Inc., "Core J2EE Patterns - Data Access Object." [Online].

Available: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html. [Accessed: 21-Dec-2011].

[70] C. Kerstiens, "Why Postgres Part 2," 2012. [Online]. Available: http://craigkerstiens.com/2012/05/07/why-postgres-part-2/. [Accessed: 31-May-2012].

[71] L. Richardson and S. Ruby, *Restful Web Services*. O'Reilly Media, 2007, p. 454.

[72] A. Hunt and D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999, p. 352.

[73] A. Reelsen, *Play Framework Cookbook [Paperback]*. Packt Publishing, 2011, p. 292.

[74] "Backbone.js." [Online]. Available: http://backbonejs.org/. [Accessed: 20-Jun-2012].

[75] J. Burke, "Require.js a JavaScript Module Loader." [Online]. Available: http://requirejs.org/. [Accessed: 24-Jun-2012].

[76] J. Creamer and nettuts+, "Key Principles of Maintainable JavaScript," 2012. [Online]. Available: http://net.tutsplus.com/tutorials/javascript-ajax/principles-of-maintainable-javascript/. [Accessed: 24-Jun-2012].

[77] The XMPP Standards Foundation, "The XMPP Standards Foundation." [Online]. Available: http://xmpp.org/. [Accessed: 20-Jun-2012].

[78] OWASP, "The Open Web Application Security Project." [Online]. Available: https://www.owasp.org/index.php/Main_Page. [Accessed: 21-Jun-2012].

[79] Twitter, "twitter developers Security Best Practices," 2011. [Online]. Available: https://dev.twitter.com/docs/security-best-practices. [Accessed: 21-Jun-2012].

[80] J. Bille, "Cobertura test coverage." [Online]. Available: http://www.playframework.org/modules/cobertura. [Accessed: 29-May-2012].

[81] Apache Software Foundation, "Apache JMeter™." [Online]. Available: http://jmeter.apache.org/. [Accessed: 23-Apr-2012].

[82] pHealth, "9th International Workshop on Wearable, Micro and Nano Technologies for the Personalized Health," 2012. [Online]. Available: http://www.phealth2012.com/. [Accessed: 26-Jun-2012].

[83] D. Guardado, "IPNlas - Presentation at pHealth." [Online]. Available: http://las.ipn.pt/home/index.php?option=com_content&view=article&id=94:presentation-at-phealth&catid=1:latest-news&Itemid=59. [Accessed: 06-Jun-2012].

[84] ORACLE-BASE, "ORACLE-BASE - PIVOT and UNPIVOT Operators in Oracle Database 11g Release 1." [Online]. Available: http://www.oracle-base.com/articles/11g/pivot-and-unpivot-operators-11gr1.php.

[85] D. Ercoli, "Drools Rule Engine Overview and Real Example." 2008.

[86] Coley Consulting, "MoSCoW Prioritisation method." [Online]. Available: http://www.coleyconsulting.co.uk/moscow.htm. [Accessed: 16-Dec-2011].

[87] *RESTful Java with Jax-RS*. O'Reilly Media; 1 edition, 2009, p. 320.

[88] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software [Hardcover]*. Addison-Wesley Professional; 1 edition, 1994, p. 416.

[89] webdesigner depot, "20 Excellent Free Rich-Text Editors." [Online]. Available: http://www.webdesignerdepot.com/2008/12/20-excellent-free-rich-text-editors/. [Accessed: 19-Jun-2012].