IPMAIA

Instituto Politécnico da Maia

Documento de síntese

Sistema electrónico de votação

Mário Ricardo de Novais Henriques

2021

Especialidade em Ciências Informáticas

Mário Ricardo de Novais Henriques

2021

Resumo

Ricardo Henriques licenciou-se em Matemática e Ciências de Computação, pela Universidade do Minho, em 1996. Em Janeiro de 97 integrou a equipa de Museus do centro de apoio ao desenvolvimento, do projecto Geira. Com financiamento Interreg, numa parceria liderada pelas Universidades do Minho e de Trás-os-Montes e Alto Douro. Em 2004 concluiu o mestrado em informática, pela mesma universidade, na área de especialização em sistemas distribuídos, comunicações por computador e arquitectura de computadores. Integra o INESCTEC[1] desde 2000, onde é investigador auxiliar no Centro de Sistemas de Informação e Computação Gráfica. As suas áreas de investigação abrangem arquitecturas de sistemas e desenvolvimento de sistemas de informação geoespaciais, baseados em normas do OGC (*Open Geospatial Consortium*), desenho e implementação de serviços e aplicações em ambientes remotos, com implementação no âmbito de projectos europeus e nacionais (em destaque, eCAALYX, ICT4Depression, eCompared, StopDepression). *Full-stack developer* em múltiplos projectos, entre os quais o sistema electrónico de votação da Ordem dos Arquitectos. Tem desenvolvido projectos relevantes em instituições e empresas, como a Sociedade Porto2001, Metro do Porto S.A., CDOS Porto, CM-Maia, PT-INOV, entre outras.

[*PT*] Este documento integra o pedido, submetido ao Instituto Politécnico da Maia (IPMAIA), de atribuição do título de Especialista em Ciências Informáticas (código 481). O trabalho de natureza profissional seleccionado, corresponde à implementação do projecto eVote - sistema electrónico de votação, o qual serviu na eleição dos órgãos directivos nacionais e regionais, para o triénio 2020 a 2022, da Ordem dos Arquitectos. Vão ser sintetizadas as tarefas desenvolvidas, abordados os desafios e a forma de execução do processo de desenvolvimento, destacando-se a intervenção do autor, em cada uma delas, e nas decisões aí envolvidas.
[*EN*] This document is part of the request, submitted to the Polytechnic Institute of Maia (IPMAIA), for the title of Computer Science Specialist (code 481). The professional work selected, corresponds to the implementation of the project eVote - electronic voting system, which served in the Order of Architects election of national and regional governing bodies, for the triennium 2020 to 2022. The developed tasks will be summarized, challenges addressed, and the executing development process documented, highlighting

[1] Instituição então designada por, INESCPorto.

the intervention of the author, in each of them, and in the decisions there involved.

Conteúdo

1 Introdução **8**
- 1.1 Organização do documento . . . 9
- 1.2 Requisitos funcionais . . . 9
- 1.3 Requisitos técnicos . . . 10
  - 1.3.1 Fichas de Inquéritos . . . 11
  - 1.3.2 eVote - Sistema Electrónico de Votação . . . 13

2 Desenvolvimento **19**
- 2.1 Metodologia de desenvolvimento . . . 20
- 2.2 Trabalho colaborativo e boas práticas . . . 22
- 2.3 Passagem ao modelo relacional . . . 28
- 2.4 Programação: inquéritos & eVote . . . 32
  - 2.4.1 Arquitectura do sistema . . . 32
  - 2.4.2 O padrão de desenvolvimento *MVC* . . . 34
  - 2.4.3 *Microsoft Entity Framework* . . . 36
  - 2.4.4 O modelo de autenticação . . . 38
- 2.5 Ligação ao serviço SMS da NOS . . . 42
- 2.6 Contagem/recepção de votos . . . 46
  - 2.6.1 REST API . . . 47
  - 2.6.2 Implementação do serviço . . . 48
  - 2.6.3 Implementação do cliente . . . 53

3 Segurança & Invariantes **55**
- 3.1 A urna é legal? . . . 56
  - 3.1.1 Arquitectura física e lógica . . . 57
  - 3.1.2 Protocolos *internet*: a utilização de *layers* . . . 58
- 3.2 Apenas eleitores autorizados podem votar . . . 60
  - 3.2.1 *One-Time Signatures* . . . 62
- 3.3 Ninguém pode votar mais do que uma vez . . . 64
- 3.4 Ninguém consegue determinar o voto de outrem . . . 67
- 3.5 Ninguém consegue duplicar o voto de outrem . . . 70

3.6 Ninguém consegue alterar o voto de outrem, sem ser descoberto 70
3.7 O voto de cada eleitor é levado a contagem e tabulação 70

4 Discussão e Resultados **73**
4.1 Sobre a metodologia 73
4.2 Sobre boas práticas 76
4.3 Sobre o *Entity Framework* 77
4.4 A votação: processo e resultados 82
4.5 Outras notas 84
4.5.1 Os *open-spaces* 84
4.5.2 O *acordo* ortográfico 84

5 Conclusões **86**

Lista de Figuras

1.1 Inquéritos: diagrama entidades-relações. 12
1.2 eVote: entidades-relações 1/2. 16
1.3 eVote: entidades-relações 2/2. 17

2.1 Estrutura de menus. 19
2.2 XP: ciclo de vida. 20
2.3 XP: decomposição em *tasks*. 21
2.4 Regressão linear (imagem Wikipedia). 25
2.5 Testes de integração. 26
2.6 Abordagem de integração: *big bang*. 27
2.7 Diagrama do contexto inicial. 33
2.8 Diagrama de voto e contagem. 33
2.9 Vista parcial do *solution explorer, Visual Studio*. 35
2.10 A arquitectura de acesso aos dados por *Entity Framework*. . . 37
2.11 Utilização do LINQ na construção do *index* de eleições. 38
2.12 Modelo de autenticação - eVote. 41
2.13 Exemplo: referência ao serviço NOS. 43
2.14 Exemplo: invocação ao serviço NOS através da biblioteca. . . 44
2.15 Exemplo: definição do *singleton*. 45
2.16 Exemplo: iniciação preguiçosa do *singleton*. 45
2.17 Exemplo: abertura e tabulação do voto. 47
2.18 Modelo de maturidade REST (RMM). 49
2.19 WebAPI: controlador e rotas (*routes*) 51
2.20 WebAPI: controlador e método. 52
2.21 WebAPI: Web.config. 53
2.22 Invocação de serviço de contagem. 54

3.1 Interface entre níveis. 58
3.2 Encapsulamento entre níveis. 59
3.3 Exemplos do cálculo do *hash* de PINs. 62
3.4 Exemplos com 3 *salt-bytes*. 64
3.5 Exemplo de um *update record*. 66

3.6 Consulta ao diário 66
3.7 Abertura do eVoto, método: *DecifraBytes2String* 68
3.8 Parâmetros RSA 68
3.9 Método: *RSADecrypt* 69
3.10 Método: *RicConverteBytes2String* 69
3.11 Contagem votos 72

4.1 GitLab: vista de Abril 2019 a Abril 2020. 73
4.2 Testes de integração e aceitação: automatizada vs. manual. . . 76
4.3 EF vs. ADO.NET: operação 1. 78
4.4 EF vs. ADO.NET: operação 2. 79
4.5 EF vs. ADO.NET: operação 3. 79
4.6 EF vs. ADO.NET: operação 4. 80
4.7 EF vs. ADO.NET: operação 5 – desempenho e crescimento. . 81
4.8 EF vs. ADO.NET: operação 6 – desempenho e crescimento. . 81
4.9 Distribuição de votantes eVote. 82
4.10 Distribuição percentual (orgão nacional). 83

Lista de Tabelas

2.4 Recursos e verbos - RMM, nível 3. 50
2.5 Controladores e métodos. 51

3.1 Invariantes do voto (electrónico). 56

Acrónimos

CA *Certificate Authority*

CLR *Compensation Log Record*

CNE Comissão Nacional de Eleições

DER diagrama de entidades-relações

DSA *Digital Signature Scheme*

EF *Microsoft Entity Framework*

GUI *graphical user interface*

HCI *human-computer interaction*

HTTP *Hypertext Transfer Protocol*

INESCTEC Instituto de Engenharia de Sistemas e Computadores - Tecnologia e Ciência

ISO *International Standards Organization*

JSON *JavaScript Object Notation*

LINQ *Language Integrated Query*

LSN *Log Sequence Number*

MD5 *Message-Digest Algorithm*

MVC *Model-View-Controller*

NOS NOS - companhia de telecomunicações

NUTS Nomenclatura das Unidades Territoriais para Fins Estatísticos

OA Ordem dos Arquitectos

REST *Representational State Transfer*

RPC *Remote Procedure Call*

RSA criptosistema de chave pública *Rivest-Shamir-Adleman*

SAML *Security Assertion Markup Language*

SHA *Secure Hash Algorithms*

SMS *Short Message Service*

SOAP *Simple Object Access Protocol*

SQL *Structured Query Language*

SSL *Secure Sockets Layer*

STS *Security token service*

TDD *Test-driven development*

TLS *Transport Layer Security*

WIF *Windows Identity Foundation*

WSDL *Web Service Description Language*

XML *Extensible Markup Language*

Capítulo 1

Introdução

Prefiro os que me criticam, porque me corrigem,
aos que me elogiam, porque me corrompem.
– Santo Agostinho

A equipa de desenvolvimento recebeu com apreensão as notícias do novo gestor de projecto, obrigado a cumprir um pequeno parágrafo de contrato que o antigo gestor, entretanto saído, tinha assumido. O parágrafo em si era um projecto completo e não uma mera alínea: a implementação de uma aplicação *Web* para a realização de inquéritos e um módulo de votação electrónica, para o triénio 2020 a 2022, da Ordem dos Arquitectos (OA).

Sendo algo novo para o centro de investigação da instituição onde trabalho e confrontados com a saída, além do antigo gestor, de outro membro – estudante de doutoramento, com responsabilidade de inovação tecnológica e de desenvolvimento – pediram-me que apoiasse a equipa neste desafio. Foi nesse contexto que entrei no projecto.

De longe, parecia não muito complicado. Contudo, havia um requisito que exalava inquietação: a necessidade de conciliar os requisitos legais, com o facto da aplicação e a base de dados da votação terem de ficar nos servidores, aos quais os próprios membros dirigentes da OA tinham acesso, sendo que também eles iriam ser uma parte interessada, ao participar, candidatando-se numa lista autónoma.

Outros requisitos foram explicitados: sob forma de entrevistas e por troca de *email*, com o secretariado e membros da OA, e ainda, por via do quadro legal aprovado e publicado em Diário da República[18].

1.1 Organização do documento

Excepto quando explicitado em contrário, o desenho da arquitectura do sistema a constituição dos modelos de dados, a programação, a configuração e execução de testes unitários, testes de integração, a configuração e/ou *deployment* são da responsabilidade do autor.

No presente capítulo apresentam-se os requisitos técnicos e funcionais, a par do contexto inicial e de desenvolvimento do projecto.

No segundo capítulo apresenta-se uma síntese do desenvolvimento. Nele se incluem os passos de desenvolvimento, a par de algum trabalho de revisão da literatura, das metodologias, *software* e técnicas utilizadas.

No terceiro capítulo o foco é a segurança. Nele vai-se discursar, mas também avaliar, de que forma os requisitos da Comissão Nacional de Eleições (CNE) e do voto electrónico tentaram ser, o melhor possível, cumpridos.

No quarto capítulo tecem-se algumas reflexões, maioritariamente associadas ao projecto, mas sem excluir a experiência acumulada, com os vários perfis assumidos em diversas instituições, em particular, enquanto investigador no Instituto de Engenharia de Sistemas e Computadores - Tecnologia e Ciência (INESCTEC).

O quinto capítulo fecha com um pequeno conjunto de conclusões, e algumas pistas de evolução.

1.2 Requisitos funcionais

A visão do sistema integra vários intervenientes, os quais, de forma integrada, colaboraram na gestão do sistema de informação da OA. À data tinha-se:

- Gestão de membros - a cargo da empresa FA Consulting.
- Componente financeira - a cargo da empresa Primavera SW.
- Gestão documental - a cargo da empresa IPBrick.
- Rede e servidores - a cargo da empresa Clara.NET.
- Portal - a cargo do INESCTEC.

A componente de inquéritos pretendia dotar a OA com uma ferramenta expedita para a elaboração, por parte dos elementos de secretariado, de fichas de inquéritos (ou questionários). O propósito incluía, melhorar a capacidade de auscultação, das preocupações e tendências dos seus membros associados.

Tratava-se de uma componente com uma visibilidade inferior e de responsabilidade limitada.

De maior visibilidade e relevância era componente de eleições, a qual foi sujeita a uma candidatura ao SAMA e na qual se podia ler:

> *"Nesta atividade, pretende-se evoluir no processo de votação para a eleição dos órgãos da ordem. Atualmente este processo é moroso, com um elevado custo de processamento e envio de documentos aos mais de 24000 membros, tornando-se num ato difícil para o Arquitecto exercer os seus direitos. No âmbito desta acção, a OA implementará um sistema eletrónico de votação que facilitará todo o processo, reduzirá significativamente os custos de cada acto eleitoral e potenciará a democratização da participação uma vez que deixam que estar presentes restrições em termos de distância e horários de funcionamento. Por outro lado, o apuramento dos resultados passa a ser mais expedito e fiável."*

1.3 Requisitos técnicos

Nesta secção colocam-se em evidência alguns dos requisitos técnicos partilhados por ambas as aplicações, sendo em subsecções específicas detalhados os requisitos particulares às fichas de inquérito e ao sistema electrónico de votação. Ambas as componentes teriam assim:

- De serem definidas numa linguagem de programação que tire partido dos mecanismos de protecção de memória, reivindicado pelos modernos sistemas operativos[64];
- Utilizar o padrão *model-view-controller*[46] com o propósito de ter o máximo de desacoplamento ("acoplamento solto", do inglês *loose coupling*), entre componentes, dispositivos e equipas[1];
- De delegar a componente de persistência na utilização do sistema de gestão de base de dados, *Structured Query Language* (SQL) da *Microsoft*;
- De ter a camada de apresentação, para o cliente final, acessível num *browser* (PC ou *smartphone*).

[1] Infelizmente, dada a curta dimensão da equipa de desenvolvimento, na maior parte das vezes foi o mesmo elemento o responsável pelo *model*, pela *view* e pelo *controller*.

1.3.1 Fichas de Inquéritos

A formulação dos requisitos para as fichas de inquéritos foi iniciada através do envio, por parte da Ordem, de um caso concreto, tido como o mais complexo de representação no sistema de inquéritos. A acompanhar o caso exemplo, juntaram os seguintes requisitos:

- A profundidade, da árvore de questões, compreender no máximo 4 níveis;
- Cada questão poder derivar num conjunto arbitrário de opções resposta, mas sempre vocabulário controlado: sem caixas de texto livre;
- As opções poderem ser de escolha múltipla ou mutuamente exclusivas;
- Os inquéritos serem preferencialmente anónimos.

Sobre esse documento foram feitas pequenas iterações com a participação de elementos da OA, outras vezes através de troca de *emails*, tendo resultado no modelo de entidades-relações expressos na figura 1.1.

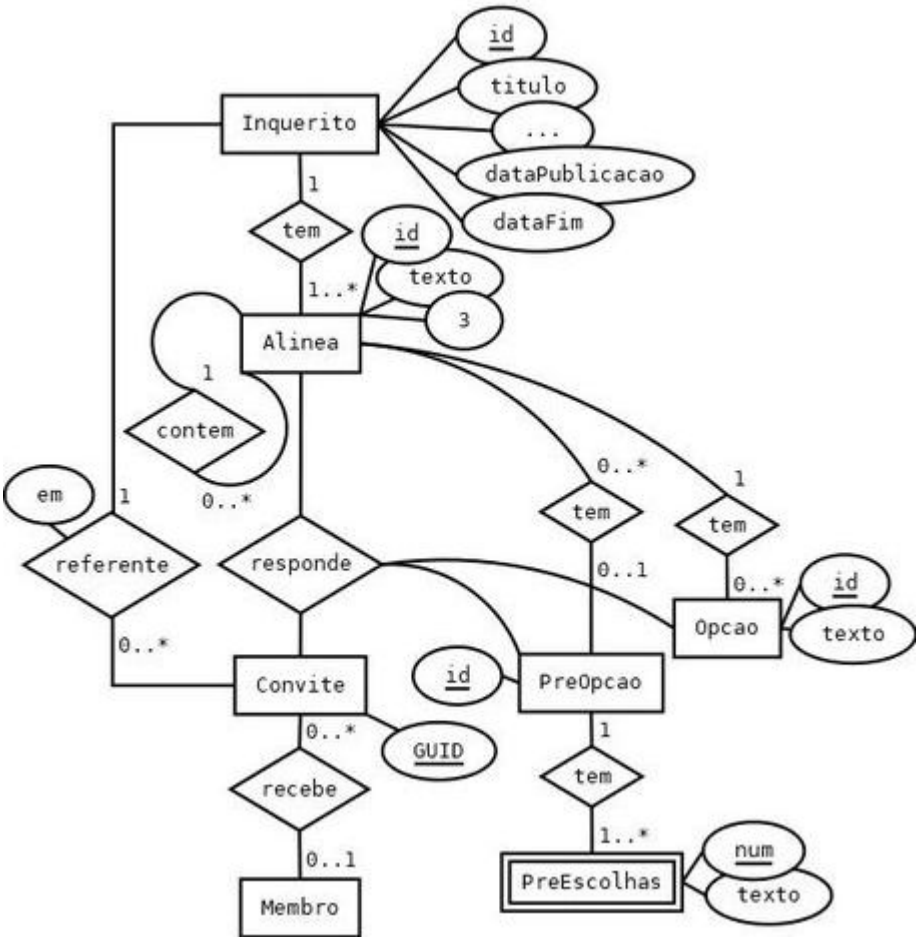

Figura 1.1: Inquéritos: diagrama entidades-relações.

Em tese a formulação deste ER[13] significa que:

- Um inquérito tem um conjunto de informação que o identifica - e que pode constar ou ser apresentada na página de preenchimento;
- Pode/deve ainda conter um conjunto de meta-informação - que sendo oculta à página que o utilizador final vê, deve reter dados sobre a sua produção, autoria, etc.;
- Um inquérito é composto (*tem*) uma ou mais alíneas;
- Uma alínea pode conter (*recursivamente*) zero ou mais alíneas;

- Uma alínea que não se desdobra noutras alíneas é uma *folha* e como tal pode conter zero ou mais *opções* - caracterizadas por um identificador e texto.

A figura 1.1 menciona ainda a noção de *pré-opção*. Ora, pelo enunciado e dúvidas colocadas, ficou claro que há um conjunto de opções tipo que podem ocorrer num ou em vários inquéritos. Por isso a para além das *opções*, as quais na prática significam – uma opção à consideração do utilizador final, para a qual o texto foi pensado especificamente para o inquérito em causa – há a possibilidade de apresentar pré-opções, com uma lista de escolhas que podem existir em múltiplas fichas de inquéritos. Assim, dando continuidade tem-se:

- Uma alínea pode, então conter um destes conjuntos pré-concebidos de *pré-opções*;
- Uma pré-opção é a composição (*tem*) um conjunto fechado de selecções, referidas como *pré-escolhas*;
- Deseja-se que os membros (todos ou uma selecção) possam aceder ao preenchimento de um inquérito através do endereçamento de um convite.

De forma muito simples, esta configuração permite ofuscar a identidade do membro através da referência ao convite; tal resulta porque, numa operação atómica[26], aquando da recepção do preenchimento ao inquérito, o convite é eliminado: deixa de poder ser usado, e ao mesmo tempo perde-se a noção de a quem foi atribuído.

1.3.2 eVote - Sistema Electrónico de Votação

A par da especificação e desenho da solução para as fichas de inquérito, iniciou-se um conjunto de iterações com a OA para compreender: os requisitos legais, funcionais, o esboço dos menus de apresentação e *interface* com o utilizador final. Nesta fase, o novo gestor de projecto, admitia que o modelo de dados das fichas de inquérito e do eVote teriam um representação semelhante/reaproveitável.

Uma diferença funcional distinguia desde logo o eVote. Enquanto as fichas de inquérito seria algo com presença apenas digital, no processo de votação abrange o eVote, totalmente digital, e a votação por correspondência. Assim, um dos primeiros requisitos seria, que a componente do eVote iria ser também responsável por contar e tabular, na fase final do processo eleitoral, os votos

em papel, mediante a introdução dos valores, por parte da comissão eleitoral. Uma outra diferença que ficou em evidência, foi a recursividade presente, em exclusivo, na definição do modelo das fichas de inquérito.

Um processo iterativo, com peripécias, avanços e recuos na compreensão e desmistificação do significado de termos e conceitos, plasmados no quadro legal, mas até pelos membros da OA, de difícil consenso. Foi preciso compreender a semântica associada a conceitos como, *zonas*, *secções*, quantas mesas de voto, a Nomenclatura das Unidades Territoriais para Fins Estatísticos (NUTS) e a associação dos membros, a forma de identificação de cada membro, número de comissões, os membros que pertencem a cada comissão, a noção de lista, os boletins e tantos outros detalhes, que não permitiam o avanço rápido.

A adicionar a estes detalhes de negócio, havia: uma equipa pequena, onde cada elemento com vários projectos distintos a correr em paralelo, e a gestão de projecto a adoptar de forma explícita o modelo incremental, modelo clássico de desenvolvimento de *software*[58], ao passo que a equipa de desenvolvimento com tiques, práticas e ferramentas adoptadas dos métodos ágeis[34].

Quando finalmente foi possível ter um modelo consolidado, a componente relativa ao modelo de dados e o resultante entidades-relações colocaram a nu que as fichas de inquéritos e o eVote tinham uma estrutura intrínseca distinta.

Tal como acima, deixa-se uma descrição dos conceitos essenciais e com reflexo no modelo ER.

- Um elemento central à organização e credibilidade do processo eleitoral é a presença do caderno eleitoral;
- Nesse caderno eleitoral constam membros, os quais podem também estar presentes em anteriores ou futuros, cadernos eleitorais, enquanto membros válidos na Ordem;
- Só podem votar na eleição os membros que constam no caderno eleitoral;
- Alguns membros fazem parte da comissão eleitoral, a qual preside e audita o processo eleitoral;
- Cada eleição contém um ou mais boletins de voto;
- Cada boletim de voto corresponde à composição das listas a órgãos (que mais não são do que opções na perspectiva do eleitor do eVote);

- Assim, uma opção num boletim corresponde a uma lista – não existindo lugar a listas uninominais;
- Um boletim corresponde à eleição de um órgão: há órgãos nacionais e órgãos *referentes* a NUTS II - correspondentes à noção de secção;
- Um membro ao *integrar* uma lista, fá-lo na qualidade de um cargo que pretende representar num órgão;
- Um membro apenas pode candidatar-se a órgãos nacionais, ou aos órgãos da secção onde está inscrito;
- O voto electrónico ($eVote$), de um membro, corresponde a um conjunto de escolhas de listas em cada uma das opções do boletim/órgão ao qual pode votar: por exemplo, um membro do norte não pode eleger órgãos da região autónoma dos Açores.

Um dos domínios essenciais de qualquer plataforma de voto electrónico é a segurança. Neste caso concreto, temos duas perspectivas que se intersectam: uma com as recomendações gerais, qualquer que seja o tipo de eleição, emanada da CNE. Outra com os requisitos que um protocolo de voto electrónico deve garantir para poder ser usado em eleições gerais.

A CNE apresenta-nos:

- Verificação Individual – a qual permite validar se o seu voto foi incluído no resultado global final;
- Verificação Universal – permite aos observadores validar se o resultado da eleição corresponde aos votos expressos;
- Verificação da Elegibilidade – permite ao eleitor e aos observadores validar se cada voto da eleição advém de um único eleitor registado (e válido).

Em [11] Schneier define os seis requisitos que o protocolo de voto electrónico deve garantir:

1. Apenas eleitores autorizados podem votar;
2. Ninguém pode votar mais do que uma vez;
3. Ninguém consegue determinar o voto de outrem;
4. Ninguém consegue duplicar o voto de outrem;

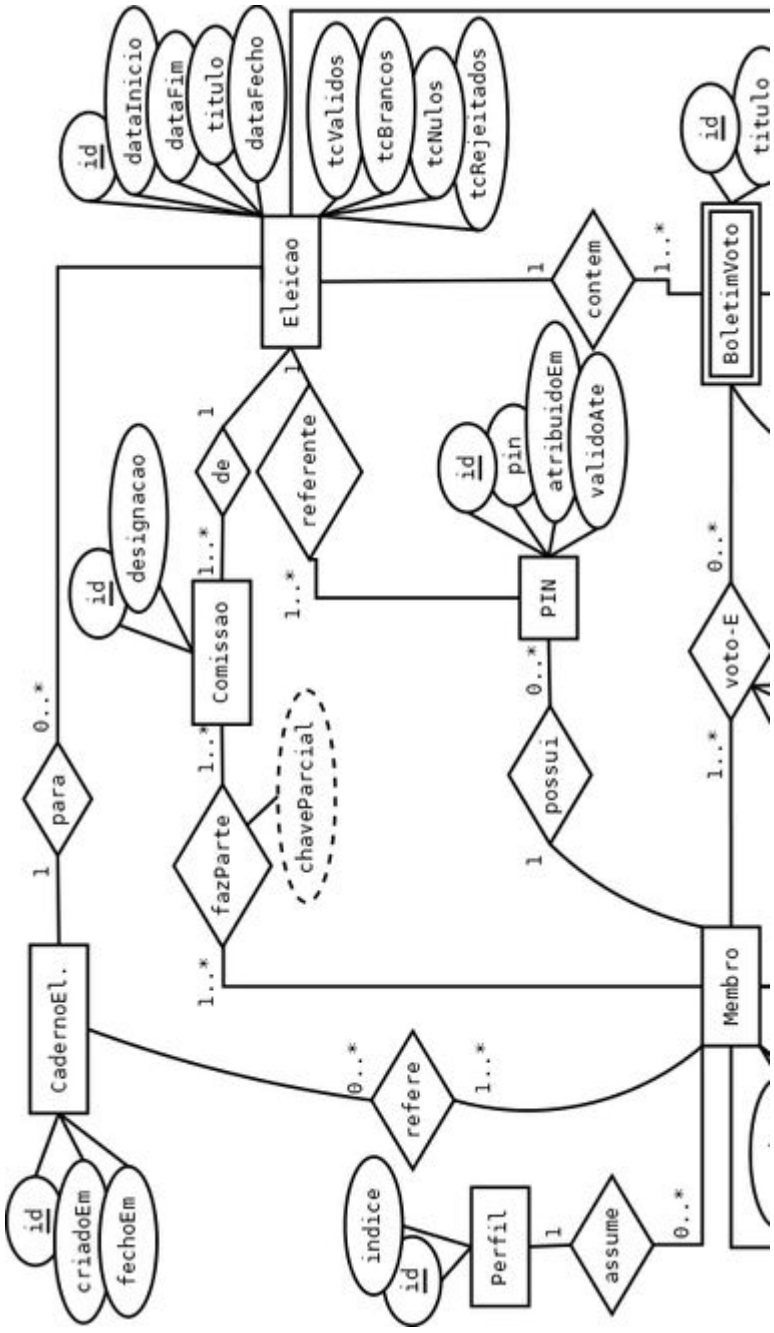

Figura 1.2: eVote: entidades-relações 1/2.

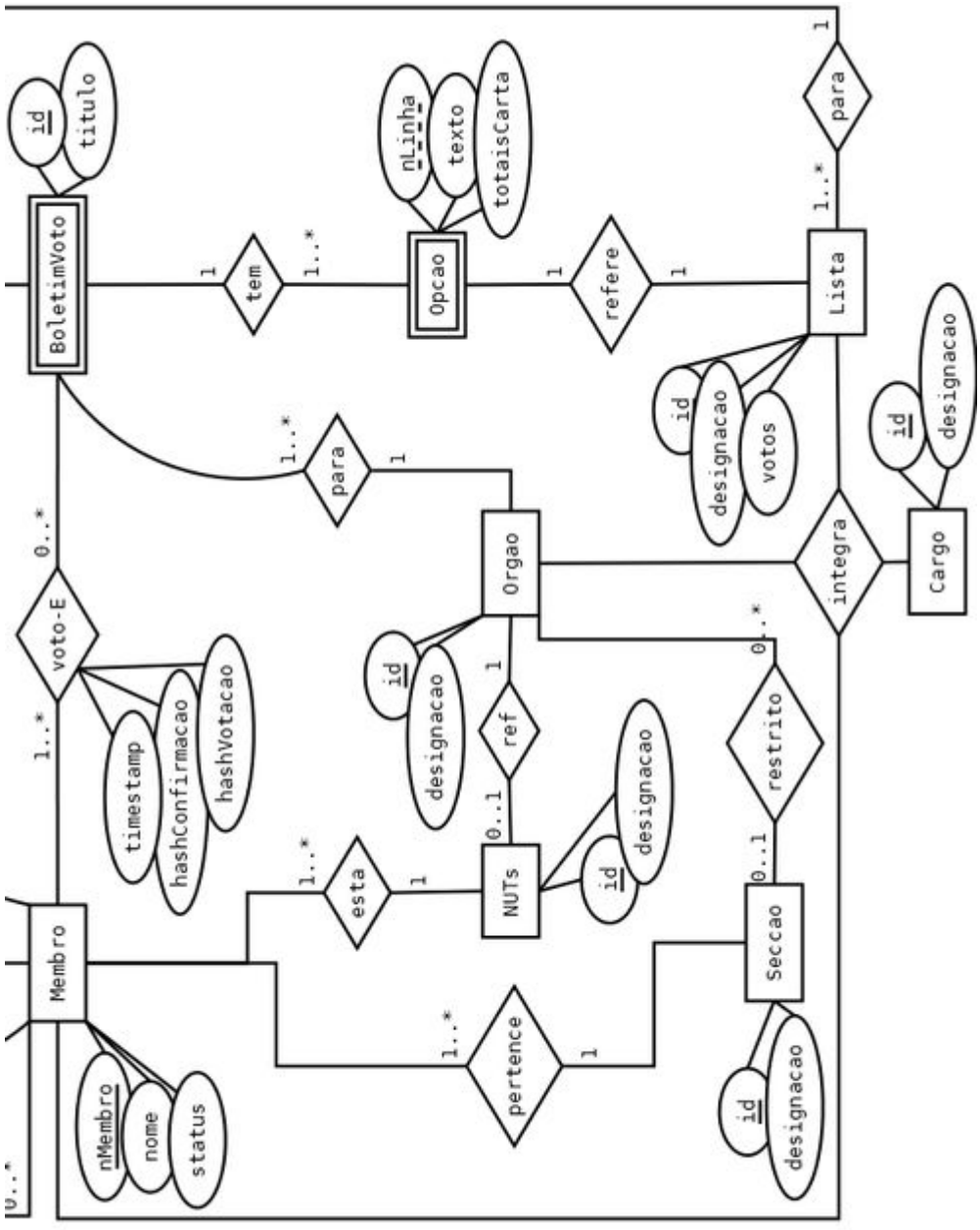

Figura 1.3: eVote: entidades-relações 2/2.

5. Ninguém consegue alterar o voto de outro, sem ser descoberto;

6. A cada eleitor que o seu voto é levado a contagem e tabulação.

Regressar-se-á a estes requisitos para analisar quão bem e se todos foram considerados e suficientemente implementados.

Capítulo 2

Desenvolvimento

Onde há alturas, há precipícios.
– Séneca

As fichas de inquérito e o eVote inserem-se numa parte do portal da OA. A figura 2.1 apresenta a hierarquia e o contexto da funcionalidade no portal, na qual, os elementos numerados de 1 a 14 foram desenvolvidos pelo autor, do presente documento. Os demais itens foram desenvolvidos pelos dois outros elementos da equipa de desenvolvimento. Uma das maiores dificuldades foi tentar ser *ágil*. Desde logo, porque a forma como a gestão apresentou de forma rígida o seccionamento do conteúdo das entregas, até ao facto de que os elementos da equipa de desenvolvimento, quase sempre, desenvolverem o código a uma só mão, parecia não deixar espaço para tal.

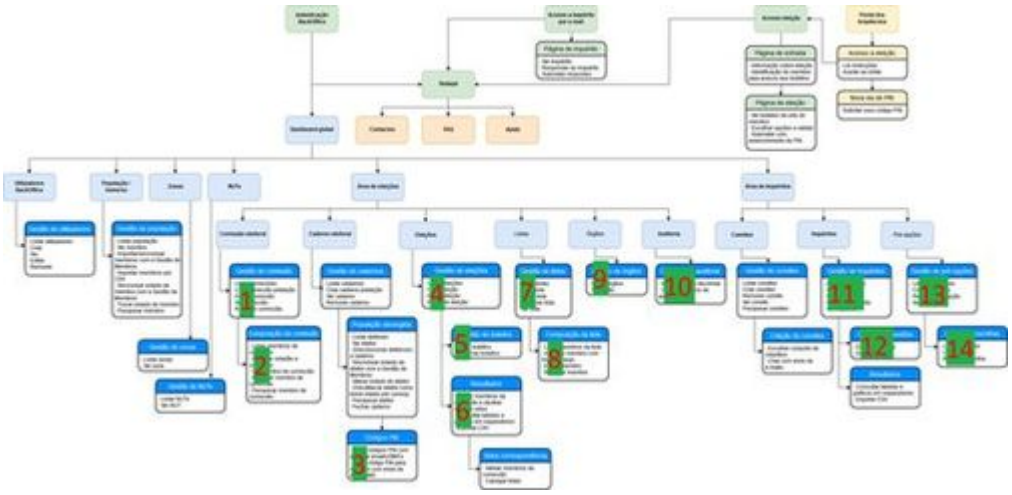

Figura 2.1: Estrutura de menus.

2.1 Metodologia de desenvolvimento

O elevado grau de incerteza e a necessidade de iniciar o desenvolvimento de parte da funcionalidade – mesmo enquanto ainda se discutia e auscultava a OA, para compreender e dominar os requisitos do sistema – apelavam à utilização de uma metodologia ágil e madura como por exemplo, o *eXtreme Programming* (XP)[39].

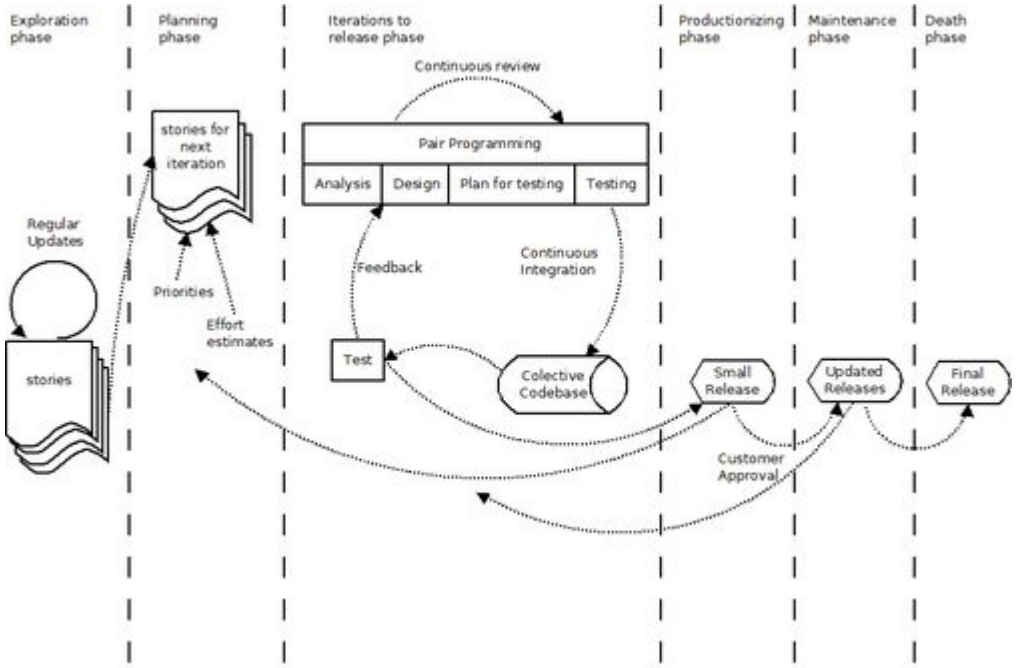

Figura 2.2: XP: ciclo de vida.

O ciclo de vida[6] do XP, representada na figura 2.2, ajuda a acompanhar a descrição da síntese das manobras com as quais se tentou, internamente, abordar o projecto de forma ágil sem a impor para o exterior nem transpirar para a gestão:

1. A *exploration phase* da metodologia ágil, dada a instabilidade dos requisitos, foi facilmente mimetizada e este processo de colectar as narrativas (*user stories*) e de as ir aprofundando, claramente ajudando a própria OA a convergir num conjunto bem definido e com os requisitos concisos; apenas se perdia no recomendado contacto mais próximo que o XP preconiza.

2. Para a *planning phase* teve-se, numa primeira instância, a selecção das narrativas que a equipa de desenvolvimento avaliou necessárias para a

constituição do esqueleto da solução final. Quanto ao escalonamento da escolha das outras *user stories* as mesmas advinham dos compromissos assumido pela gestão, para a funcionalidade a disponibilizar em cada versão, do modelo incremental. Aqui teve-se uma das razões do sucesso do projecto. O modelo incremental assume que os requisitos estão fechados e bem definidos, estabelecendo versões em que sucessivos requisitos vão sendo fechados. Ora, a gestão do projecto não tinha tal, mas achava que tinha. De facto não se tratavam de requisitos, mas antes nomes de módulos (que mais não eram do que caixas negras). Como o desenvolvimento por detrás estava preparado para esta incerteza de requisitos e com as *user stories* ainda a sofrerem completação, esta fase de planeamento ágil permitiu uma rápida mudança de prioridades e uma estimativa de esforço mais assertiva.

3. As ditas *iterations to release phase* foram parcialmente implementadas de forma ágil. Por exemplo, tudo que tinha componentes computacionais mais complexas seguiam o caminho de análise, desenho, preparação e programação dos testes unitários, implementação e validação. Usava-se também a integração contínua, mas apenas uma a duas vezes por dia. Como se indicou, infelizmente o código não passava por várias mãos, sendo ainda impensável aplicar *pair programming* – algo observado como improdutivo pela gestão. Os *sprints* nem sempre foram homogéneos, tendo em média sido de quatro semanas.

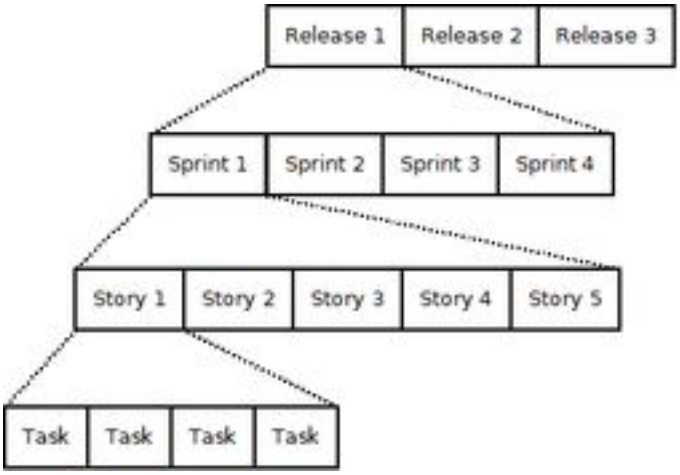

Figura 2.3: XP: decomposição em *tasks*.

4. Foram feitas 2 *releases* para as fichas de inquéritos e 5 *releases* para o eVote. Assim a *productionizing phase* das metodologias ágeis foram

evoluindo para ir completando os ditos módulos funcionais com os requisitos do cliente. Dessa forma a gestão nem se apercebeu que a agilidade subjacente foi preponderante para o cumprimento atempado dos objectivos. No caso do eVote, e tal como se espera no XP, houve um natural encurtamento das iterações, dado que eram mais pormenores com reflexo na interacção homem-máquina do que componentes novos que exigissem grande desenvolvimento.

5. Finalmente a *maintenance phase* supõe que o produto esteja em uso e com suporte, e assim foi, mas não se colocou a continuação do desenvolvimento de novas iterações, pelo que saltou para a...

6. *Death phase* assim que os resultados foram publicados e o processo eleitoral fechado.

2.2 Trabalho colaborativo e boas práticas

Ao dispor do projecto, e configurado pelo serviço de administração de sistemas, tínhamos o sobejamente conhecido GitLab[67]. Das muitas funcionalidades que dispõe, o seu uso no projecto foi, maioritariamente, para:

- Partilha e gestão de código fonte (*source code management*);
- Integração contínua (*continuous integration*);
- Controlo de actividades e problemas (*issue tracking*) em particular:
  - Discussão e implementação de novas ideias;
  - Controlo de tarefas e estado das actividades;
  - Dúvidas e suporte a novos pedidos.

Fosse por via de instruções na linha de comando, ora por ferramentas mais amigáveis, tais como o TortoiseGit[59], ou integrado no próprio Visual Studio, foi bem aceite a sua utilização por todos os membros da equipa de desenvolvimento.

A parte realmente difícil, e só atingida de forma parcial, foi a implementação no dia a dia, de outras boas práticas de engenharia. Sendo um investimento inicial, devolvem em mais valia: em qualidade e bom andamento do desenvolvimento [37]. Em síntese valerá a pena analisar:

- Integração contínua[24] e *check-ins* mais frequentes – bem suportado pelo GitLab, se o código passasse por mais mãos em simultâneo teria de ser feito mais do que apenas uma a duas vezes por dia. Esta boa prática foi a mais amplamente usada. Com ela, rapidamente a equipa se apercebia:
  - Se algo produzido por um afectou a compilação de outro parceiro;
  - Se algo causou algum efeito disruptivo nos testes;
  - Do estado global da iteração, ao percebermos até onde cada elemento tinha conseguido evoluir.
- Implementação do modelo *test-drive development* – apenas um terço da equipa de desenvolvimento o utilizou e, ainda assim, apenas nas situações cujo desenvolvimento se adivinhava menos linear. Ao contrário do que o nome possa induzir, *Test-driven development* (TDD) não é um processo de teste, mas uma forma de desenhar *software* onde o código é desenvolvido em ciclos curtos: vermelho, verde, reestruturar. Mas o que testar? A mnemónica *Right-BICEP*[30] aponta 6 áreas concretas, onde estão a generalidade dos problemas:
  1. *Right* - os resultados estão certos? Se enquanto programador não conseguir responder satisfatoriamente a esta questão, então escrever o código (ou o teste) é uma total perda de tempo! Isto não vai contra a agilidade? Então e se os requisitos forem vagos ou incompletos? Então não se pode escrever código se os requisitos não estiverem completamente definidos? Se os requisitos não forem completamente conhecidos ou completos, como programador deve/pode assumir algo como verdade e testa sobre essa assumpção. Ou seja, pode não estar correto sob o ponto de vista do utilizador final, mas pelo menos o próprio programador tem de saber o que o código deve fazer: se não sabe o que testar não sabe o que programar!
  2. *B* - as condições de fronteira (*boundary*), são bem conhecidas? Aqui é importante ter bem presente o que se espera – se está conforme; se está dentro do intervalo válido; se refere algo externo, então existe e tem acesso; se apresenta a cardinalidade correcta e ainda se os eventos estão a ocorrer na ordem esperada, no tempo certo, e a tempo?
  3. *I* - consegue testar relações inversas (*inverse*)? Sobre alguns métodos é possível aplicar o seu inverso matemático ou lógico o que

permitirá uma mais forte validação. Outro exemplo é validar se um valor foi inserido com sucesso se ele passou a existir na tabela da base de dados. Em qualquer dos casos: cautela. Às vezes algum vício de raciocínio pode estar afetar ambos, o método original e o seu inverso. Recomenda o bom senso, onde possível, usar uma fonte diferente (que não as classes do próprio programador) na validação pelo método inverso.

4. C - consegue validar através de outras formas o resultado, *cross-check*? O caso anterior pode ser visto como um caso particular deste. Quando não for possível aplicar o inverso, ou se podendo aplicar uma validação como a anterior, desejar reforçar o teste, poderá optar por um *cross-check*. Um caso muito comum é durante o processo de refinamento do código o programador surgir com uma nova implementação com melhor desempenho, ou que utilize menos recursos. Uma forma fácil então de fazer o *cross-check* é validar se os resultados da implementação nova são iguais aos da primeira implementação (antes também validada, obviamente)! Outra situação trivial, é por exemplo: se estiver a codificar um programa para gestão de inventário, de vez em quando deve fazer a contagem física, para confirmar se o que a aplicação conta em *stock*, corresponde à realidade.
5. *E* - consegue forçar o surgimento de situações de erro, (*error conditions*)? No mundo real as situações de erro ou inesperadas ocorrem: discos enchem, perdem-se as ligações à rede e os sistemas operativos falham. O ideal é que se consigam testar as reações do seu código, o comportamento das suas classes, no evento de situações dessas. Esta área foi sobejamente usada para a abertura e contagem de votos – tanto mais se testa quanto maior é o medo!
6. *P* - o desempenho, a *performance*, está dentro do esperado? Um domínio onde pode ser benéfico examinar são as características de desempenho - não o desempenho em si, mas as tendências que se observam à medida que o tamanho do *input* aumenta, ou à medida que os problemas são mais complexos. Pode ser útil arquivar um rápido teste de regressão das característica de desempenho.

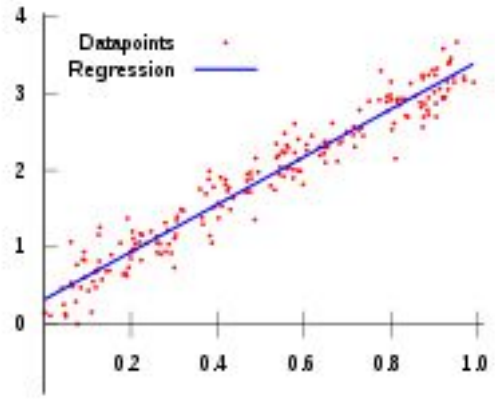

Figura 2.4: Regressão linear (imagem Wikipedia).

- O *refactoring* – as empresas não a cultivam, não investem na sua formação, não valorizam quem a pratica. O *refactoring*, ou em português, refinamento ou reestruturação, não tem a sua pouca utilização na ausência de valor, mas antes na falta de cultura dos gestores das empresas e instituições. Transmitida a percepção acumulada pela vida, dos muitos anos de programação, muitos perfis diferentes de clientes e de gestores, vamos ao sumo. Refinar passa por compreender quais os factores que compõem a solução actual. Desde cedo, ainda em criança, foi-nos ensinada a importância de decompor um problema, seja informático ou não, para que se identifiquem unidades mais pequenas do que precisa de ser resolvido ou programado – para de forma mais fácil raciocinar uma solução, enquanto dividida em subproblemas.

  Quase sempre, a necessidade de *refactoring* é sentida, quando surge a necessidade de introduzir uma nova funcionalidade, mas o código do programa não está estruturado de uma forma conveniente para essa adição de funcionalidade. O programador sente isso e deve reestruturar o programa para tornar mais fácil essa adição: primeiro reestruture-se e depois adicione-se a funcionalidade!

  Qualquer alteração de código, mesmo que se julgue para melhor, é sempre uma edição de código e, portanto, uma fonte natural de introdução de erros (*bugs*). Como tal reestruturar sem a presença prévia de testes, é deixar o conforto da ciência para enveredar pelo campo da crença...

  Um ponto muito relevante de Kent Beck em [8] é a metáfora dos dois chapéus. Tentei incorporar esse conceito; porém por vezes são manhosos os caminhos da mente; enquanto o cérebro está ciente dos princípios

e se garante a abstração de refinar apenas ou codificar apenas, a produtividade – ou a sua percepção – torna-se mais gratificante.

Os dois grandes subconjuntos em que se sentiu a necessidade de *refactoring* foram, na presença de código duplicado e na presença de grandes (extensos) métodos. Abaixo uma breve lista dos padrões de *refactoring* usados:

1. Código duplicado
   - *Extract method*
   - *Pull up field*
   - *Form template method*
   - *Substitute algorithm*
2. Métodos grandes
   - *Extract method*
   - *Replace temp with query*
   - *Introduce parameter object*
   - *Preserve whole object*
   - *Replace method with method object*

- Integração automática e testes de aceitação – os testes de integração surgem quando partes distintas do sistema se unem para colaborar. Duas partes ou módulos de um sistema são integrados, sendo relevante validar essa integração.

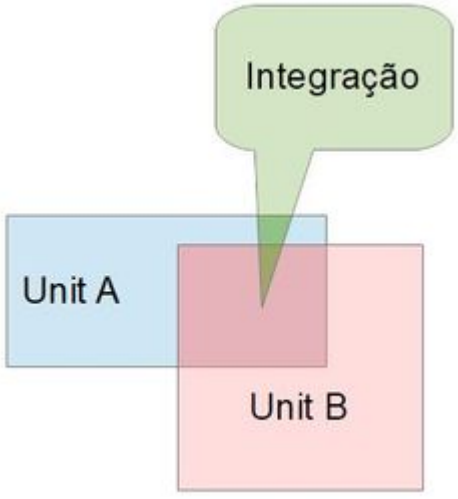

Figura 2.5: Testes de integração.

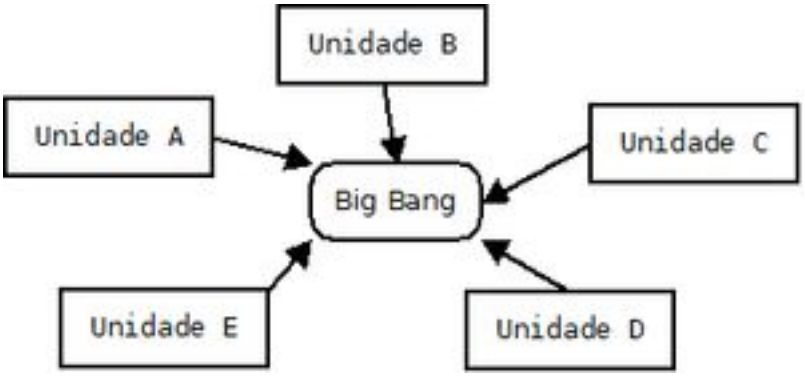

Figura 2.6: Abordagem de integração: *big bang*.

Estes testes devem incidir sobre módulos que previamente foram, unitariamente testados, com sucesso. O teste de integração define se a composição dos módulos resultou como se esperava, ou não.

No projecto a forma de integração usada seguiu a abordagem *big bang*, figura 2.6.

Nesta abordagem, todos os módulos, classes ou unidades são integradas e testadas de uma só vez. Normalmente usa-se esta abordagem quando todo o sistema está pronto para fazer a integração.

Note-se porém que esta abordagem continua a ser um teste de integração, pois apenas se testa se os módulos, classes e unidades integram, sem se estar a fazer testes a todo o sistema como um todo.

Do ponto de vista da gestão, esta abordagem é interessante ao permitir juntar as equipas que desenvolveram as unidades numa fase em que todos sincronizam para garantir a integração.

- *Pair programming* – em síntese a programação em pares traduz-se em dois membros da equipa desenvolverem em comum sobre a mesma tarefa: um teclado, dois cérebros. Sendo uma prática preconizada pelo XP e várias outras abordagens ágeis, não foi usada. Um dos pontos negativos, que o seu não uso pôs em evidência, é a insuficiente sobreposição de conhecimento na equipa, pelo que a parte de um dificilmente é reconhecida e com maior dificuldade será enriquecida se for necessário, por outro.

Passe-se então à descrição de um dos primeiros passos do desenvolvimento.

2.3 Passagem ao modelo relacional

A modelação através de diagrama de entidades-relações (DER) não se tratou de uma opção pessoal[1], mas antes por reunir as características da facilidade para abstracção e comunicação entre os elementos, da equipa de desenvolvimento e gestão, na qual se cruzaram escolas diferentes, sendo os DER um denominador comum.

O primeiro passo passou por aplicar as regras[19] de derivação que permitem derivar o modelo relacional. Neste modelo conceptual:

- Faculta-se uma forma de descrever entidades e relações entre elas;
- Todas as entidades e relações são descritas como tabelas;
- Cada tabela é um conjunto de registos (linhas da tabela), todos semelhantes no seu tamanho e formato;
- Cada registo é composto por diversos campos, atributos, que correspondem às colunas da tabela.

Para a representação de uma tabela deve-se indicar:

- O nome da tabela;
- Os campos chave;
- Os outros atributos.

Cada tabela pode ser derivada seguindo regras simples, desde que se garanta o cumprimento, sequencial, da análise ao DER . Primeira etapa apenas tomando a definição isolada das entidades:

[1] Utilizaria VDM-SL[38].

Uma entidade no DER	Uma tabela no MR
• Atributos da entidade	• Atributos da tabela; • Atributo chave (se existir) torna-se parte da chave primária; • Uma tabela representa uma entidade (ou classe); • Cada registo da tabela representa uma instância dessa entidade; • No caso de atributos compostos, cada parte do atributo gera um atributo diferente na tabela; • Atributos multi-valor: agir como se o atributo fosse uma nova entidade relacionada com a anterior através de uma relação de 1:M.
• Entidades fracas	• Derivar uma tabela para esta entidade como fosse uma entidade normal; • Juntar à chave os atributos da chave da entidade dominante; • A relação identificadora torna-se redundante – se tiver atributos podem ser colocados na entidade fraca.

• Generalização - há duas formas de representar:	1. Deriva-se uma tabela G para a classe genérica seguindo as regras apresentadas; faz-se uma tabela para cada subclasse seguindo as regras normais, juntando a cada uma os campos correspondentes à chave primária de G. 2. Ou não se defina a tabela para G e em cada subclasse replicam-se os atributos de G.

Na segunda etapa o foco da conversão centra-se nas relações com mais de duas entidades e relações M:N.

Uma relação destas no DER	Produzirá sempre uma nova tabela no MR
• Relação M:N	• Estas relações são sempre representadas numa nova tabela; • Cada linha da nova tabela tem de identificar uma entidade de cada um dos lados da relação – incluindo as chaves de cada um dos lados da relação. • A tabela deve ainda incluir todos os atributos da relação; • A chave é constituída pelas chaves das duas tabelas ligadas.

• Relações com mais de 2 entidades (grau de aridade superior)	• Estas relações são sempre representadas numa nova tabela; • Cada linha da tabela inclui as chaves de cada entidade presente na relação relação, mais os atributos da relação; • Se a relação for de cardinalidade *muitos* para todas as entidades, todos os atributos são parte da chave. • Se alguma entidade tiver cardinalidade *um* na relação, então o número de chaves candidatas é igual ao número de cardinalidades de *um*.

Na terceira etapa o foco da conversão centra-se nas relações 1:M e 1:1.

Uma relação destas no DER	Produzirá normalmente, no MR uma referência da entidade *um* na entidade *muitos*[2]
• Relação 1:M	• Relações entre entidades A e B, onde cada elemento de A apenas pode relacionar-se com uma entidade de B; • Um elemento de A define um elemento de B: A tem uma *referência* para B; • Incluir na tabela A os campos da chave de B.

[2]Se a chave for formada por muitos campos ou se a relação tiver atributos pode ser mais fácil derivar uma nova tabela com a chave de A, a chave de B e os atributos da relação.

• Relação 1:1	• A chave da relação é apenas a chave de A; • Se a cardinalidade de B for *[1..1]* os campos da chave estrangeira não poderão ser nulos; • Se a cardinalidade de B for *[0..1]* a chave estrangeira pode permitir valores nulos; • Não se consegue garantir a não opcionalidade do lado A (ou a relação fica vazia).

2.4 Programação: inquéritos & eVote

Os tópicos cobertos nesta secção, em termos de revisão da literatura e em termos do paradigma de desenvolvimento, abrangem a componente das fichas de inquéritos e do sistema electrónico de votação. Nas secções 1.3.1, 1.3.2 apresentaram-se os respectivos modelos e em 2.3 sintetizaram-se as regras que levaram à definição dos modelos relacionais, vertidos na implementação do sistema de base de dados relacional SQLSERVER da *Microsoft*.

Nas próximas secções vai-se abordar como, um conjunto de abstrações e mecanismos colocados ao dispor no IDE, foram usados no âmbito do desenvolvimento das aplicações: fichas de inquéritos e eVote.

2.4.1 Arquitectura do sistema

A arquitectura do sistema, inicialmente pensada, segue uma simples e tradicional partição de sistemas. O repositório de informação, suportado por um sistema de base de dados relacional, e a componente aplicacional suportada pelo servidor IIS, para albergar as aplicações *web* de inquéritos e eVote.

Esta visão ignorava como se iriam superar alguns desafios, em particular a limitação indicada logo na introdução deste documento, da base de dados da votação ficar acessível, em servidores, aos quais os membros dirigentes da OA tinham acesso. A esperança da equipa, nos primeiros meses, era de poder contar com a colaboração de um especialista em segurança, que nos orientasse nesse e noutros desafios.

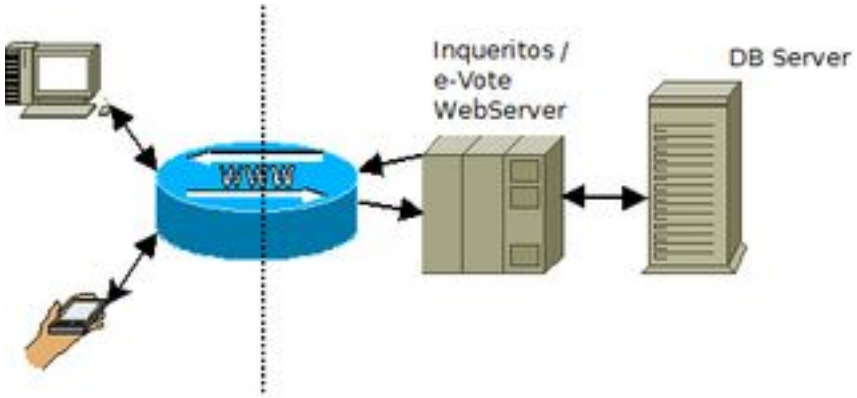

Figura 2.7: Diagrama do contexto inicial.

A figura 2.8 mostra um diagrama simplificado dos principais processos e mensagens na arquitectura final. Pode-se observar, que além da fronteira entre os sistemas dos clientes e o sistema de voto electrónico, acrescentou-se à arquitectura de 2.7 uma nova fronteira e um novo sistema específico, o qual ficou com a responsabilidade absoluta de proceder à decifragem dos votos e a sua contagem.

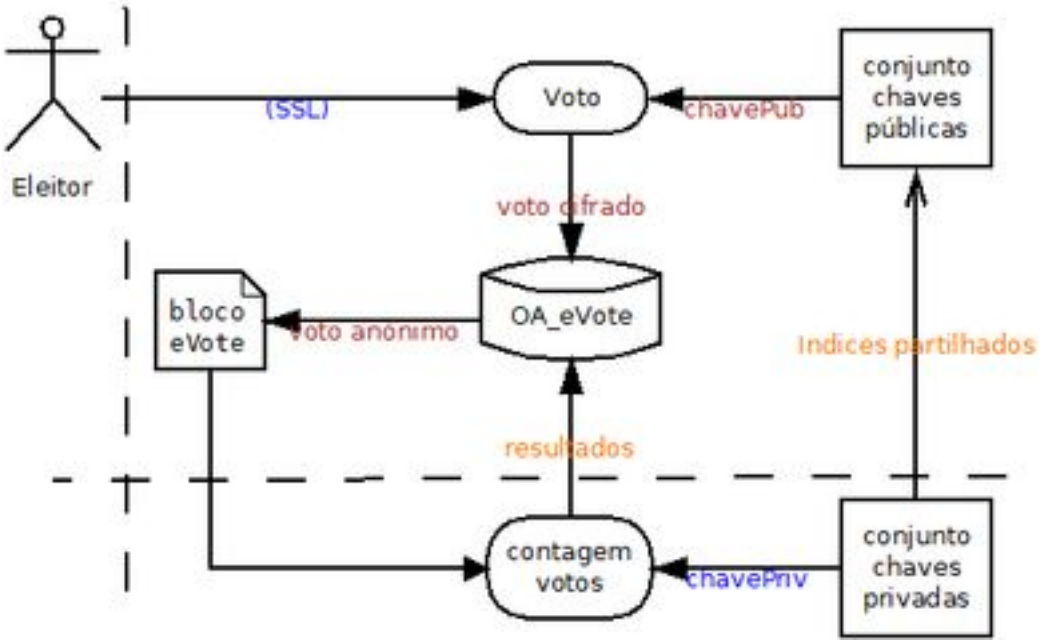

Figura 2.8: Diagrama de voto e contagem.

É importante partilhar com o leitor, que a definição da arquitectura e

todo enquadramento do desenvolvimento, assentou num princípio de maximizar o uso de componentes ou bibliotecas disponíveis, no ambiente de desenvolvimento, complementando, ou blindando as áreas mais expostas ou vulneráveis.

A introdução desse novo sistema foi essencial em termos de legitimidade do eVote, ao proteger o anonimato do eleitor, por detrás de cada voto, e ainda, por permitir aferir que cada voto foi levado a contagem e tabulação; dois dos seis requisitos fundamentais, enunciados em 1.3.2.

2.4.2 O padrão de desenvolvimento *MVC*

Já em 1998 Royce[50] indicava que o produto técnico mais importante de um projecto de *software* é a sua arquitectura. Do ponto de vista da gestão, a arquitectura é apenas um conceito de desenho do *software*, na qual se incluem, a engenharia (de *software*) necessária para a completa elencagem dos sistemas a adquirir, os produtos de desenvolvimento, as comunicações e a quantificação em número dos recursos humanos a envolver. Incluem ainda a elaboração dos custos de desenvolvimento individual de cada componente.

Para a estimativa do custo de desenvolvimento de cada componente, poder ser passado à gestão, a equipa de desenvolvimento já tinha alguma informação necessária, com base no levantamento de requisitos e na formulação do modelo relacional. O modelo relacional é, ele próprio, uma reificação sobre um dos eixos: o dos dados. O outro eixo, das operações – ou funcionalidade, num sentido menos estrito – não estava definido, excepto no que respeitava a alguns esboços (*mockups*) da *graphical user interface* (GUI) para o utilizador final.

Com a pressão da gestão para estimar os valores e com base no tipo de *software* que se visava desenvolver, estava-se ciente que o mais célere seria adoptar uma arquitectura *software* robusta, em detrimento da formulação de uma qualquer nova. Tanto melhor estando a mesma bem suportada pelo ambiente de desenvolvimento especificado. Assim, adoptou-se o *Model-View-Controller* (MVC)[52], como padrão. Ao mesmo tempo isso ajudou a lidar com a incerteza, ao dividir as fronteiras da funcionalidade, tal como preconizado neste padrão, e assim esboçar um intervalo com a estimativa de tempo. No centro da incerteza tinha-se uma caixa negra: a segurança. Iremos dissecá-la no capítulo 3.

Em poucas palavras, porque é bem conhecido, o MVC é um padrão de desenho de *software*, muito utilizado em sistemas com presença de GUI[3].

[3]Enquanto estudante-viajante, chamávamos na gíria, de sistemas *click-oriented*... provavelmente (estatizações à parte) hoje designar-se-iam de *tap-oriented*!☺

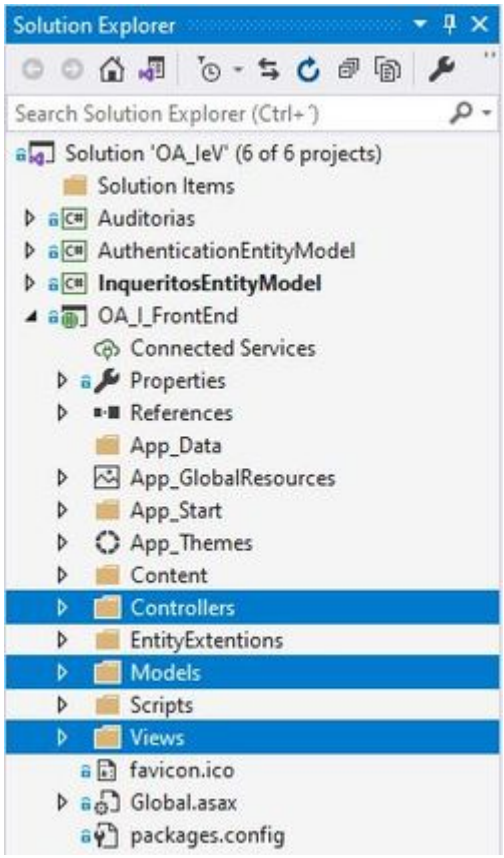

Figura 2.9: Vista parcial do *solution explorer, Visual Studio*.

Apresenta uma divisão lógica em três componentes interligadas:

- O *Model* deve reter a lógica (regras) de negócio, as quais permitem a validação e alteração do estado dos objectos.
- A *View* retém a lógica de apresentação; múltiplas formas de apresentação podem coexistir, com base na mesma informação.
- O *Controller* é responsável intermediar os *inputs* do utilizador e encaminhar os dados para o *model*, ou em resposta receber os dados do *model* e orientar a *view* na preparação da apresentação/resposta.

A programação ao nível de controladores e modelos foram codificadas em C# e a apresentação em *Razor* – potenciadas, com alguma validação lógica no lado do cliente, através de *jQuery*.

A figura 2.9 coloca em evidência as pastas correspondentes onde se definiram as vistas, os modelos e os controladores do MVC. A mesma figura exibe o projecto *InqueritosEntityModel*, o que nos levará à próxima secção.

Como nunca tinha desenvolvido em *Razor* os primeiros exemplos pareciam simples; mas assim que era necessário garantir a passagem de valores do modelo para a apresentação, ou vice-versa, surgiam as primeiras dúvidas. A primeira parte a ser codificada prendia-se com a *interface* que permite, ao secretariado da OA, definir a estrutura em árvore de alíneas ou subquestões. O código não ficou bonito, mas cumpre. A parte seguinte foi a componente da *interface* com o utilizador final enquanto membro, ou seja, com o perfil de quem irá preencher/responder ao questionário. O entendimento e o à-vontade melhorou, nomeadamente, perdendo-se o medo de adaptar e estender os modelos ou desenvolver novos.

2.4.3 *Microsoft Entity Framework*

A motivação para a utilização da *Microsoft Entity Framework* (EF) prende-se com a sua própria definição: trata-se de uma biblioteca nativa do .NET, a qual coloca ao dispor do programador, uma estrutura de computação capaz de garantir o mapeamento bidireccional do modelo relacional para o nível de abstracção de classes/objectos, da camada lógica de negócio. Assim, a EF permite que o desenvolvimento se faça a um nível superior de abstração na codificação de aplicações, o que resulta na diminuição e simplificação do código produzido. No caso do eVote, servia ainda de denominador comum, uma vez que o *background* dos membros da equipa de desenvolvimento, na plataforma ADO.NET, era heterogéneo.

A EF não pretende ser apenas uma solução de mapeamento entre objectos e o modelo relacional. Permite às aplicações acederem e alterarem os dados que são representados como entidades e relações no modelo relacional e usa a informação no modelo e nos ficheiros de mapeamento para traduzir em interrogações sobre objectos, em interrogações dos tipos representados no modelo conceptual e desses em interrogações específicas ao nível do repositório de dados. Por fim, os resultados de tais interrogações são materializados em objectos geridos pela EF (figura 2.10).

De acordo com a documentação da *Microsoft*[40] a EF faculta duas formas de colocar interrogações ao modelo conceptual e devolve objectos:

- LINQ para entidades - faculta um suporte para, através da *Language Integrated Query*[2], interrogar tipos de entidades que estão definidas no modelo conceptual.
- *Entity SQL* - faculta um dialecto de SQL para aceder directamente às entidades do modelo conceptual com suporte para as classes do *Entity Data Model*; pode ser usado através de consultas de objectos ou por execução de interrogações acessíveis via *EntityClient*.

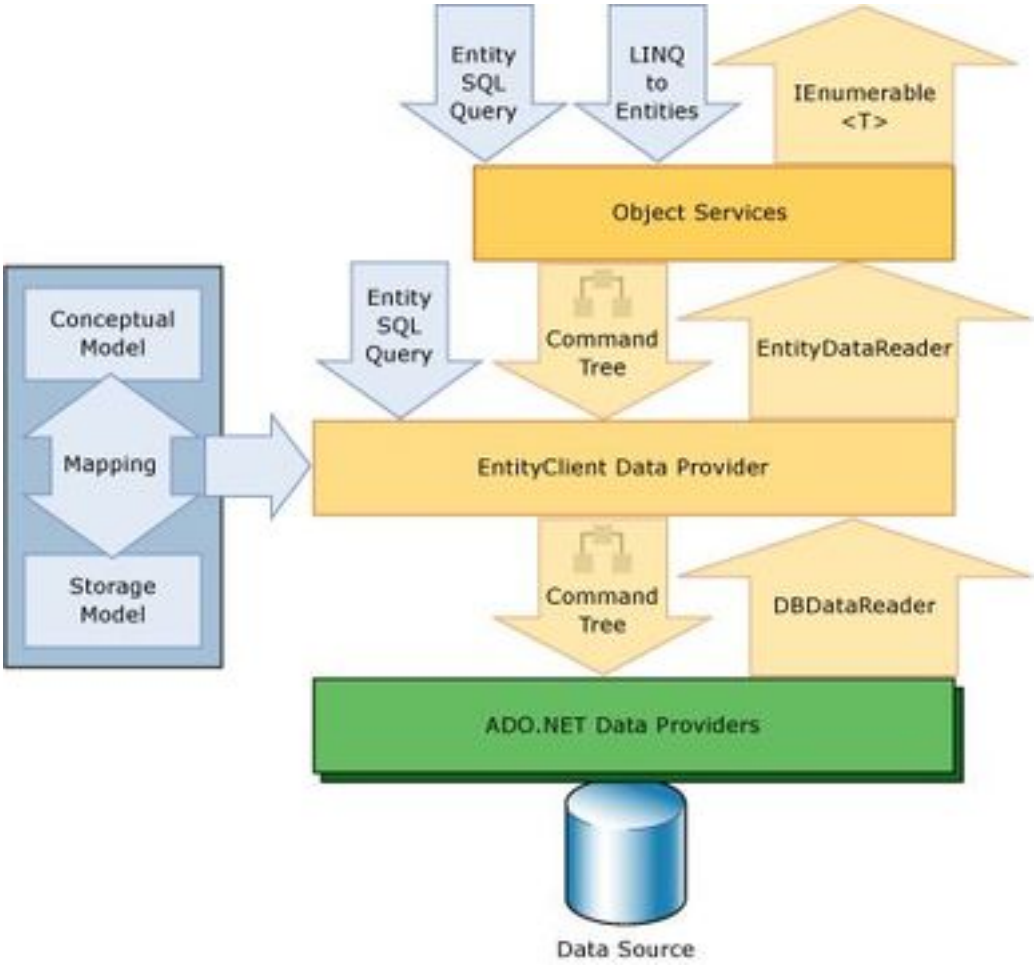

Figura 2.10: A arquitectura de acesso aos dados por *Entity Framework*.

No repositório de informação retinham-se os modelos base correspondentes ao modelo de dados transposto para a base de dados. Ao dispor no C#, estava a encapsulação dessas estruturas em objectos da EF e *Language Integrated Query* (LINQ)[2].

```
public async Task<ActionResult> Index() {
    var view = new EleicaoListaViewModel();
    DateTime hoje = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day);
    var _eleicoesActivas =
        (from e in db.OA_Eleicao_Eleicoes
            select new EleicoesActivas {
                RefEleicao = e.idEleicao,
                TituloEleicao = e.titulo,
                DataEleicao = e.dataInicioVoto,
                ListasPorEleicao = (from l in db.OA_Eleicao_Lista
                                    where l.refEleicao == e.idEleicao
                                    orderby l.designacao
                                    select new ListaEleitoral { IdLista = l.idLista,
                                        Designacao = l.designacao,
                                        Mote = l.mote })
                                    .ToList()
            }).OrderByDescending(x => x.DataEleicao);
    view.Eleicoes = await _eleicoesActivas.ToListAsync();
    return View(view);
}
```

Figura 2.11: Utilização do LINQ na construção do *index* de eleições.

Em síntese, as *frameworks* de mapeamento objectos/relacional (O/RM) facultam formas convenientes de obter a abstração do modelo de negócio e das suas regras, sem preocupação do acesso aos dados. A EF, ao dispor e incentivada pela *Microsoft*, deixou a sua mais valia na facilidade de gerar código de forma mais expedita. Porém, o compromisso desta abstração, pretere o desempenho[57]. Na secção 4.3 deixa-se uma breve análise comparativa.

2.4.4 O modelo de autenticação

À semelhança da formação recebida, de tantos outros programadores da mesma geração, o tradicional currículo académico ignorava alguns domínios, hoje omnipresentes, como o caso do controlo e gestão da autenticação e autorização de acessos a uma aplicação ou sistema. Tal necessidade de controlo de acessos era uma preocupação traduzida pela OA, naturalmente, sob forma de um requisito. No contexto de desenvolvimento e de acordo com o princípio enunciado em 2.4.1, foi-se à procura de uma solução mais confortável para lidar com os conceitos de autorização e autenticação, para evitar ter de programar de raiz tais artefactos.

A qualquer programador familiarizado com programação ASP.NET, WCF (*Windows Communication Foundation*) ou que desenvolva aplicações *Web*,

surje[4] ou é proposta a utilização do *Windows Identity Foundation* (WIF)[10], muitas vezes apelidado simplesmente por *identity*[41], e disponível, pelo menos, desde a *.NET framework version 3.5 SP1*.

O WIF compreende um conjunto de classes do .NET para implementar o designado *claims-based identity model*[27]. Como a designação inglesa deixa entender (*claim* $\simeq$ *alegar*), passa por confiar num sistema de identificação externa, o qual está configurado para facultar às nossas aplicações o conhecimento necessário sobre o utilizador, a par das necessárias garantias de cifragem, e de que os dados recebidos provém de uma fonte fidedigna. Sob este modelo, é fácil com um simples *single sign-on* permitir o acesso à aplicação eVote, sem que esta tivesse a seu cargo as seguintes e importantes tarefas:

- Autenticação de utilizadores.
- Armazenamento de contas e credenciais.
- Conectar aos directórios (às *enterprise directories*) para pesquisar detalhes de identidade do utilizador.
- Garantir a integração com sistemas de identidade de outras plataformas ou instituições.

Assim, sob este modelo, a aplicação eVote toma decisões relativas à identidade com base nas alegações facultadas pelo utilizador. Um dos maiores benefícios da utilização do *identity* passa por permitir que as equipas de desenvolvimento possam focar-se na implementação da lógica de negócio.

Passe-se de imediato à revisão e alguns termos e conceitos disponíveis na literatura *Microsoft*.

Identity

Pese embora o termo *identity* apareça acima e na literatura como algo que abarca conceitos como a autenticação, autorização, de facto, para o programador, podemos olhar para o termo como um conjunto de atributos que descrevem um utilizador, ou entidade, com o qual estamos interessados em trabalhar, do ponto de vista da segurança.

Claim

O *claim* é um pedaço de informação da identidade, tal como um nome, uma idade, ou um endereço de *email*. Quantos mais *claims* a aplicação receber

[4] Às vezes no próprio *wizard* de iniciação das soluções.

mais informação do utilizador ficará no seu âmbito. A razão semântica deste termo advém, de que neste modelo, a aplicação não irá recolher os atributos num directório, mas é antes o próprio utilizador que deixa estes atributos na aplicação. Portanto a aplicação recebe-os, mas com um determinado grau de incerteza.

Security Token

No caso de uma aplicação como o eVote, por se tratar de uma aplicação *Web*, os ditos *claims* são recebidos juntamente com o pedido num *POST HTTP*, a partir do *browser* do utilizador; e podem inclusive, serem posteriormente armazenados como *cookie*, se for solicitada uma sessão. À parte do protocolo sob o qual chegam, os *claims* precisam de ser serializados e, é neste contexto, que surge o *security token* (evidência/prova de segurança). Um *security token* é um conjunto de *claims* assinados digitalmente pela entidade emissora. Tal assinatura confere que o utilizador não usurpou de uma identidade pela qual se fez passar.

Uma das características de fundo do WIF passa por prestar de forma transparente a criação de leitura dos *security tokens* entregando ao programador valores passíveis de leitura.

Issuing Authority

Uma *issuing authority*, ou entidade emissora, podem ser de múltiplas classes, desde um governo, banco, ou ainda controladores de domínio, ou autoridades de certificação[65]. No caso específico em análise, trata-se tipicamente de um *web service* ou aplicação que conhece como emitir os referidos *security tokens*. A referida *issuing authority* é um agente central na componente de identificação, pois é nela que se confia, por procuração, para identificar o utilizador.

Security Token Service

Este serviço é responsável por encaminhar os *security tokens* de acordo com protocolos interoperáveis.[5] O WIF implementa e abstrai as dificuldades e o encargo da implementação destes protocolos. Se houver necessidade de construir na solução um *security token service*, a API do WIF permite fazê-lo, sendo necessário apenas definir as regras (*security policy*) que o definem.

[5]Neles se incluem as especificações *WS-Trust* e *WS-Policy*.

Relying Party

De acordo com a literatura, quando se constrói uma aplicação que confia em alegações (*claims*), está-se a desenvolver uma *relying party*, ou também designada como *claims-aware application*[6]. No caso concreto, como se indicou desde logo na secção 1.2, a gestão dos membros, no que respeita a inscrições, actualização e pagamento de quotas, recai sobre outra empresa, na qual o nosso sistema, no papel de *relying party* confia, e nos serve enquanto *authority*.

Standards

Para que este modelo seja interoperável vários *standards* WS-* entram em jogo[53]. As convenções (*policy*) são consultadas via GET HTTP e obtém um resultado estruturado pela especificação *WS-Policy*. O *Security token service* (STS) expõe interfaces que implementam a especificação *WS-Trust*, os quais descrevem como o pedir e receber os *security tokens*. Uma forma comum de emitir (entregar) os *security tokens* é através da *Security Assertion Markup Language* (SAML).

No eVote...

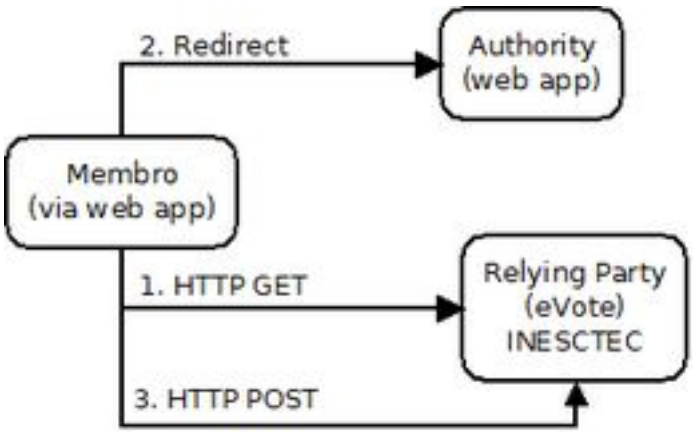

Figura 2.12: Modelo de autenticação - eVote.

A figura 2.12 elucida a situação onde o membro, através do seu *browser* pode participar neste cenário de autenticação. Trata-se de um cenário simples onde a aplicação no cliente é designada também como *passive client*. Através

[6] Ou *claim-based application*.

do *browser* o membro refere a *claim-aware web application*, o eVote. Esta, por seu turno, redirecciona o *browser* para o STS, para que o membro possa ser autenticado. O STS lê o pedido com a solicitação, autentica o membro através de mecanismos usuais de HTTP, cria o SAML com o *security token*, e emite *javascript* para induzir o *browser* a iniciar um HTTP POST que, por fim, envia o *security token* ao *relying party*.

2.5 Ligação ao serviço SMS da NOS

Nesta secção descreve-se de forma sintética, a criação de uma biblioteca para encapsular a ligação ao serviço de entrega de *Short Message Service* (SMS) da NOS - companhia de telecomunicações (NOS), via *Simple Object Access Protocol* (SOAP). A utilização deste serviço ocorre, na aplicação eVote, no momento anterior à colocação das escolhas eleitorais do membro. A mesma biblioteca foi também colocada ao serviço de uma *desktop application*, do secretariado da OA, para facilitar o envio de SMS, e como meio complementar de ligação aos seus associados.

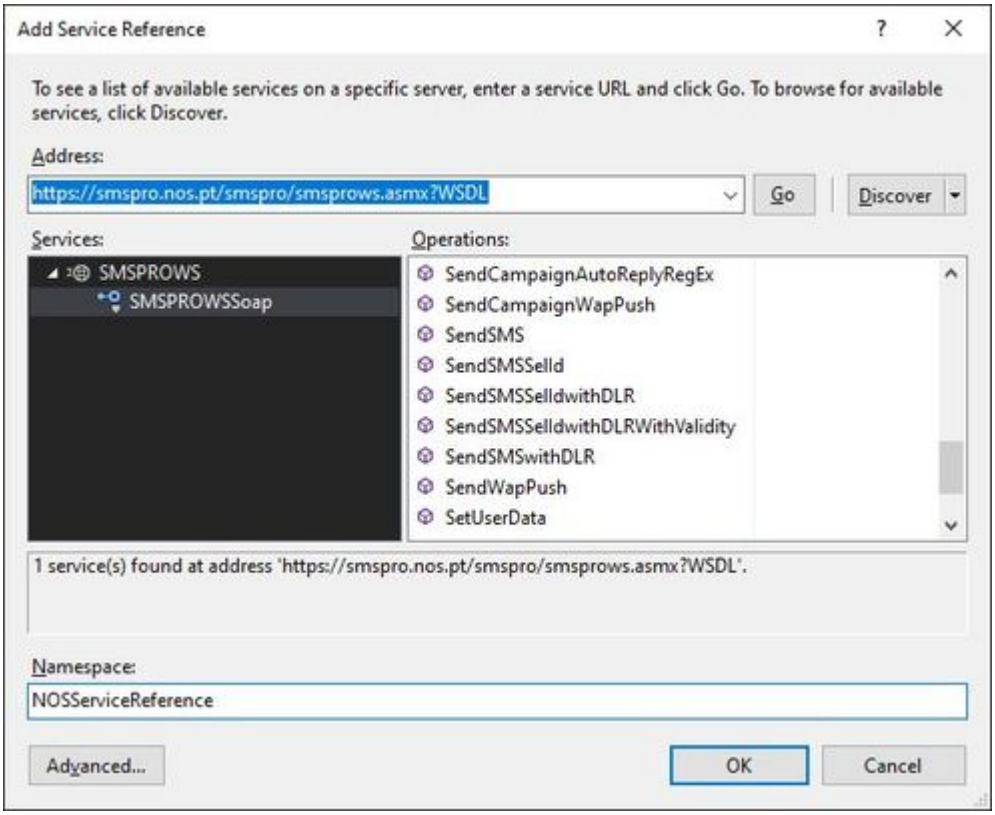

Figura 2.13: Exemplo: referência ao serviço NOS.

Em https://smspro.nos.pt/smspro/smsprows.asmx?WSDL pode-se consultar a descrição dos métodos disponíveis, na *Web Service Description Language* (WSDL)[7], ainda sob os termos da definição *WSDL v1.1*. A versão 1.2 (renomeada 2.0) tem como principal adição, vincular todos os métodos HTTP, para além do GET e POST; à data da escrita deste documento mantinha-se com a definição original.

As duas formas mais comuns, de colocar a nossa aplicação como cliente de um serviço SOAP passam por, ou tirar partido do Visual Studio, como se exibe na figura 2.13, ou mediante a criação de *stubs*, com intermédio, por exemplo, do aplicativo de desenvolvimento[8] .NET, *svcutil.exe*. A segunda alternativa foi a usada, com os seguintes argumentos:

```
svcutil.exe /language:cs /out:GeneratedProxy.cs
/config:App.config
https://smspro.nos.pt/smspro/smsprows.asmx?WSDL
```

[7] Ou também designado por *Web Service Definition Language*
[8] Em *Developer Command Prompt for Visual Studio 2019*.

Foi ainda necessário adicionar as seguintes referências à classe criada:

```
System.Runtime.Serialization
System.Runtime.Serialization.Formatters.Soap
System.ServiceModel
```

Para a aplicação que usa a biblioteca o esforço de enviar um SMS formatado será simples. A forma mais complicada é a invocação de um SMS formatado, como se exemplifica na figura 2.14. Ainda assim é linearmente trivial:

```
SMSTools.Instancia.EnviarSMSFormatado(new SMSDestinatario {
    Genero=Sexo.masculino,
    NomeOpcional = "Sebastião Zeferino",
    Telefone = 910000000 },
    "Mensagem Personalizada",
    "Ordem dos Arquitectos",
    "OA-SRS", true)
```

Figura 2.14: Exemplo: invocação ao serviço NOS através da biblioteca.

Por detrás, esconde-se a utilização de um dos mais conhecidos padrões de implementação, o *singleton*. Na essência um *singleton* é uma classe que deverá permitir criar apenas uma instância de si; em adição, providencia um qualquer expediente para aceder a essa instância. Impõe-se a necessidade de garantir que o processo de instanciação está protegido (*thread-safe*). Significa isto que, a porção de código para validar que só uma instância de si existe, não é invadida (questionada) por mais de um *thread*. Caso contrário, haveria o perigo de ambos terem determinado (lido) que não havia nenhuma instância, pelo que cada *thread* iria criar a instância – ambos iriam criar, o que violaria a determinação de ser *singleton*.

Historicamente, *sempre* foi possível a utilização de apontadores para funções, com codificação tipo-C, para criar entidades designadas de *callback functions*, ou apenas *callbacks*. Através delas, o programador pode configurar uma função para reportar de volta de outra função ou aplicação. É através deste mecanismo que se dispunha de *handles* para uma acção de um botão, ou do movimento do rato, ou em geral qualquer comunicação bidireccional entre duas entidades de programação. O problema com esta codificação tipo-C, é que estes *callbacks* representam pouco mais do que um simples endereço de memória[61]. Sem incluírem o mínimo de informação quanto ao número e tipo de parâmetros, ou o tipo de resultado a devolver, estas invocações clássicas de *callbacks* eram causas de erros ou outros desastres de execução. Não obstante, *callbacks* são entidades úteis.

No .NET os *callbacks* e a sua funcionalidade está disponível, numa forma mais segura e orientada aos objectos, através dos *delegates*. Na essência é um objecto (tipo seguro) que refere (aponta) para outro método (ou lista de métodos) na aplicação, para ulterior invocação. Com o .NET3.0 houve a introdução de *lambda expressions*. Uma *lambda expressions* é um método anónimo que pode conter expressões e instruções podendo ser usadas para construir métodos *delegate*, ou *expression trees* - estas representam código numa estrutura de dados tipo árvore, onde cada nodo é uma expressão.

Dada a natureza do eVote, de se tratar de uma *web*[9] *application*, no contexto do IIS, e sem utilização de mecanismos *alive cross sessions*, a utilização do padrão *singleton*, será algo redundante, ao garantir que o mesmo utilizador final, não é responsável por múltiplas instâncias e pedidos ao serviço NOS, nem sobrecarregue o sistema eVote, cliente do serviço. Contudo, há outra parte interessante da solução implementada. Ao dispor do programador, a partir do .NET4, tem-se um mecanismo que utiliza o expediente designado de *lazy initialization* (iniciação preguiçosa), para deferir a criação de objectos, especialmente quando a criação ou execução pode nem ocorrer durante a vida útil do programa, e normalmente associado, a recursos intensivos ou pesados. Traduzindo todos estes factores em código C# tem-se:

```
private static readonly Lazy<SMSTools> preguica = new Lazy<SMSTools>(() => new SMSTools());
5 references
public static SMSTools Instancia { get => preguica.Value; }
```

Figura 2.15: Exemplo: definição do *singleton*.

Com a utilização do tipo *Lazy*, apenas é necessário passar um *delegate* ao construtor, o qual invoca o construtor do *singleton*.

```
private SMSTools() {
    lock (mute) {
        try {
            smsTenant = ConfigurationManager.AppSettings["idNOSListaSMS"];
            smsUser = ConfigurationManager.AppSettings["smsUser"];
            smsPass = ConfigurationManager.AppSettings["smsKey"];
            client = new SMSPROWSSoapClient("SMSPROWSSoap");
            if (client == null) throw new Exception("SMS Tools",
                new Exception("SMS Tools endpoint return null"));
        } catch (Exception ex) {
            throw new Exception("SMS Tools (consult inner exception details)", ex);
        }
    }
}
```

Figura 2.16: Exemplo: iniciação preguiçosa do *singleton*.

[9]Stateless por desenho.

Concluído o pedido à instância (ou a sua criação) seria apenas invocar como se exibiu na figura 2.14.

2.6 Contagem/recepção de votos

O contexto do processo de contagem de votos, surge definido num sistema à parte, de acordo com o diagrama da figura 2.8 da secção 2.4.1. O seu propósito primário é contabilizar os eVotos. Como este serviço consegue abrir e decifrar o eVoto, para garantir o anonimato do eleitor, é essencial que não receba a informação de quem é o voto. Assim, quem conhece o membro, o sistema eVote, não consegue abrir o voto. E quem consegue abrir o voto, o sistema de contagem, não recebe a informação a quem pertence a escolha. O que recebe é um conjunto de triplos de informação:

- `idBoletim` - uma simples *string* com o identificador unívoco de um boletim, o qual pertence a uma só eleição.
- `hashVoto` - um *array* de *bytes* com o voto secreto - a escolha escondida que o eleitor tomou.
- `indicePK` - o índice da correspondente chave pública, com a qual o voto foi cifrado.

Com base nesta informação vai iterar por todos os boletins recebidos. Primeiro passo é encontrar, através do índice, a chave primária. Abre o voto e preenche a tabulação. Esta é implementada com recurso a uma *hashtable*, `Dictionary` do .NET, cuja chave é composta pelo identificador do boletim, e a opção (voto após ser decifrado). O valor, correspondente a uma chave, é um número natural com o somatório de coincidências dessa chave. Sempre que a chave é nova, é adicionada uma nova entrada no dicionário e colocado o valor a 1. Se a chave já existe, apenas se procede ao incremento em uma unidade. Simples!

```
internal void DesdobraVoto(TuploVoto voto, Parameters2Save x) {
    if (contador != null && voto != null && x != null) {
        string opcaoEscolhida = RicRSA.DecifraBytes2String(voto.HashVoto, x);
        Chave c = new Chave { Boletim = voto.IdBoletim, Opcao = opcaoEscolhida };

        if (contador.ContainsKey(c)) {
            contador[c]++;
        } else {
            contador.Add(c, 1);
        }
    }
}
```

Figura 2.17: Exemplo: abertura e tabulação do voto.

A montante deste processo, de cálculo dos resultados da eleição, tem-se: um mecanismo de recepção do conjunto dos boletins – um serviço REST; o seu cliente, a plataforma eVote. Estes dois vão ser detalhados nas próximas duas secções e, no capítulo 3, ter-se-á oportunidade de apresentar o método `DecifraBytes2String`.

2.6.1 REST API

O *Representational State Transfer* (REST) foi criado por Roy Fielding[22], o qual é também, um dos autores iniciais da especificação *Hypertext Transfer Protocol* (HTTP). O REST foi pensado para tirar partido de padrões e tecnologias dentro do HTTP, numa simplicidade de abstração que suplantou, o até há poucos anos dominante SOAP[14] – na tecnologia de acesso a *web services*. Ao contrário deste último, onde um número arbitrário de métodos podiam ser criados, o REST encoraja que os programadores de APIs REST, façam uso exclusivo dos métodos `GET`, `POST`, `PUT` e `DELETE`[36]. Para tal, o REST caracteriza-se por ser centrado nos recursos (*resource-centric*). Isto significa que uma API *RESTful* utiliza os verbos HTTP para agir, ou consultar informação, sobre recursos. Na nomenclatura REST, diz-se que os recursos são os substantivos (ex., eleitor, boletim), e tem-se os métodos, verbos, a actuarem sobre os recursos. Ainda é de destacar, enquanto importante aspecto do REST, o facto de tirar partido de outros mecanismos dos sistemas HTTP, tais como:

- *Caching*
- *Security*
- *Statelessness*

- *Network layering*

Vale a pena clarificar, para que não transmita a ideia de que o SOAP não tem valor ou perdeu a sua utilidade. As capacidades do protocolo SOAP vão além das do REST, bastando, para tal, estudar as especificações WS-*[15]. Elas lidam com necessidades de comunicação mais complexas, tais como segurança, transacções, procura (descoberta) de serviços, publicação de metadados, roteamento, relações de confiança, entre outros[10]. Nenhum destes são possíveis no REST, uma vez que requerem capacidades fora do âmbito do protocolo HTTP.

Pelo REST aplicar somente *standards* HTTP, torna-o extremamente interoperável, para todas as plataformas capazes de colocarem pedidos HTTP. Tal inclui os óbvios computadores, *smartphones*, *tablets*. Mas não só: sistemas de telefones, máquinas ATM, *vending machines*. Qualquer sistema, por mais simples que seja, com capacidade de fazer pedidos HTTP, pode também *dialogar* REST. O mesmo se aplica às representações de dados em *JavaScript Object Notation* (JSON)[23][33] e *Extensible Markup Language* (XML) simples. Quando comparados com o SOAP, estas tecnologias exigem pouquíssimo, quanto à formatação ou na especificação das mensagens. Pese embora o SOAP se baseie num protocolo XML[28], o encargo de construir uma mensagem válida em SOAP, com inclusão de um envelope, um cabeçalho, e um corpo, é muito mais complexo do que comunicar apenas os próprios dados. O REST associada à própria forma de formatação minimal do JSON, revela-se como uma vantagem diferenciadora para sistemas com conexão intermitente ou por parte de instrumentos/aparelhos de baixo consumo energético, bem como para utilização sobre redes de comunicação móvel, quando comparado com a mesma representação de informação XML/SOAP.

2.6.2 Implementação do serviço

De todas as condicionantes e desafios, a construção do serviço REST para atender e contar os eVotos, foi dos passos mais fáceis e com menos impacto no calendário do projecto. O que a experiência permite transmitir, é: pode haver um esforço primordial, que tem a ver com a mudança da mentalidade, quanto à forma de pensar o serviço. Dos puros *Remote Procedure Call* (RPC), ainda em meados dos anos 90, construídos à custa de *stubs* e implementação em C, até cerca de 2014, no projecto ICT4Depression[49], com serviços SOAP, senti que, em abstracto, a diferença era mais sintáctica que semântica. Porém, quando no projecto subsequente, projecto eCompared[62][35], abracei o REST, foi necessário construir uma nova abstracção. Nessa altura sim,

[10] *Identity federation*, por exemplo.

senti o referido esforço primordial de adaptação. Contudo, isso só custa uma vez (e provavelmente, quem nunca ficou contaminado com as abstracções do passado, nem sinta qualquer dificuldade).

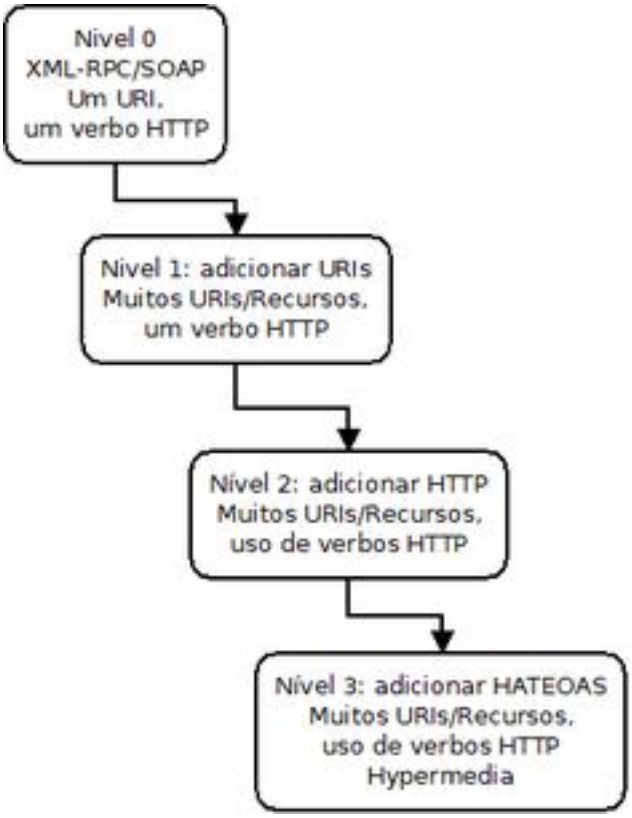

Figura 2.18: Modelo de maturidade REST (RMM).

Quem estiver reticente, ou quem pressionado, por motivos profissionais o tiver de fazer com agilidade, poderá seguir o modelo de maturidade do REST de Richardson [47] (RMM), como guia de construção e melhoria de serviços REST, como se esquematiza na figura 2.18.

A construção de uma *interface* REST impõe que se obtenha uma API dita *resource-centric*. Por isso, o mencionado esforço primordial, coloca-se na forma de pensar: intencionalmente passam a ser os recursos, os substantivos, o cerne da solução. No estilo RPC, por oposição, o serviço crescia à volta dos métodos, os quais podiam ser em número arbitrário. Aqui, cingidos, aos verbos/métodos disponíveis em HTTP, o REST fica com quatro acções para consultar e manipular os recursos. Como não se tem liberdade para criar outros verbos, o conceito central do REST, tal como o nome indica, passa a estar na inteligente definição dos recursos. Em síntese, sob forma de uma

tabela, têm-se a seguinte lógica operacional entre verbos (métodos) e um recurso (tipo ou classe de objectos):

Verbo HTTP	URI (sobre conjunto) http://serv.pt/recurso	URI (sobre um elemento) http://serv.pt/recurso/123
GET	Lista todos os elementos, dessa classe de recurso, incluindo URIs para cada elemento individual.	Obtém um elemento único, identificado pela URI.
PUT	Substitui o conjunto inteiro de elementos, dessa classe de recurso.	Substitui (edita) o elemento individual, identificado pela URI.
POST	Cria um novo elemento, dessa classe de recurso; o seu identificador é gerado pelo sistema.	Cria um novo elemento, subordinado ao identificador indicado na URI.
DELETE	Elimina todo o conjunto de elementos, dessa classe de recurso.	Elimina apenas o elemento, dessa classe de recurso, identificado pela URI.

Tabela 2.4: Recursos e verbos - RMM, nível 3.

Por fim, e a propósito de código implementado no projecto, mencione-se acerca de outros códigos, o de retorno – *HTTP status code*. Tal como o programador está constrangido a usar apenas os verbos/métodos HTTP, também deve limitar-se a devolver códigos retorno, de entre o conjunto dos códigos do *stantard* HTTP[54].

Quando em 2.4.2 se apresentou o papel do *MVC* na aplicação *web* eVote, ficou claro a importância da noção do controlador (*controller*). Também ao nível dos serviços ASP.NET, incluindo a designada *Web API*, tem como peça fundamental o controlador. De facto, na constituição do serviço REST, não se tem vistas (*views*). E o modelo (*model*) é apenas usado para guardar e mover dados dentro da aplicação. O controlador é usado pela *framework* para responder aos pedidos, e executa parte relevante da lógica que o serviço requer. Isto traduz-se no seguinte: quando um pedido *web* chega pela rede ao IIS (*Internet Information Services*), este usa as rotas configuradas na aplicação para determinar qual o controlador que deve atender o pedido. Quando a apropriada classe controlador é descoberta, a *framework MVC* cria uma instância dessa classe, e entrega o pedido ao apropriado método. No `./App_Start/WebApiConfig.cs` ficou a definição pré-definida pelo Visual Studio, da configuração de rotas de entrega:

```
namespace Conta_eVotos {
    1 reference
    public static class WebApiConfig {
        1 reference
        public static void Register(HttpConfiguration config) {

            // Web API configuration and services

            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

Figura 2.19: WebAPI: controlador e rotas (*routes*)

O mecanismo do *MVC* vai procurar corresponder à definição do mapeamento das rotas, para determinar o controlador apropriado a activar. Neste caso irá utilizar porção do URL especificado a seguir ao `api/`. Falta apenas esclarecer qual o método, que dentro do *controller* deverá tratar o pedido.

Ora, o conjunto das bibliotecas do projecto, *ASP.NET MVC 5 Web API*[11], tem uma semântica própria de associação entre o verbo/método HTTP, e o nome atribuído ao método, que detém a implementação da acção a executar.

URL	Verbo HTTP	Método no controlador
api/recurso	GET	Get()
api/recurso/123	GET	Get(long id)
api/recurso	POST	Post()
api/recurso/123	PUT	Put(long id)
api/recurso/123	DELETE	Delete(long id)

Tabela 2.5: Controladores e métodos.

[11]Desde versões anteriores.

```
[Route("api/Contagem")]
[ResponseType(typeof(Tuple<string, string, long, string>[]))]
//[ResponseType(typeof(Contagem))]
1 reference
public async Task<IHttpActionResult> PostVotos(TuploVoto[] votos) {
    Contagem c = new Contagem();
    try {
        if (votos != null) foreach (TuploVoto v in votos) {
                var x = from _privado_ in db.KeyPoolPair
                        where _privado_.idPool == v.IndicePK
                        select new Parameters2Save {
                            D = _privado_.c1,
                            DP = _privado_.c2,
                            DQ = _privado_.c3,
                            Exponent = _privado_.c4,
                            Inverse = _privado_.c5,
                            Modulus = _privado_.c6,
                            P = _privado_.c7,
                            Q = _privado_.c8
                        };
                c.DesdobraVoto(v, x.FirstOrDefault<Parameters2Save>());
            }
    } catch (Exception ex) {
        c.UltimoErro = ex.Message;
    }
    if (c != null)
        return Ok(c.Resultado());
    else
        return NotFound();
}
```

Figura 2.20: WebAPI: controlador e método.

Ao usar a *Web API*, a URL não inclui o segmento da acção. A *Web API* mapeia os métodos do controlador com base no verbo/método HTTP que o cliente colocou no pedido. Por exemplo, se for o `GET`, será invocado o `Get()` da classe do controlador, se for um `POST` invocará o método com o nome `Post()`, tal como indica a tabela 2.5. De facto, ainda é mais expedito, pois mesmo que o nome não coincida, acaba por ser deduzido, de acordo com a tabela. A imagem 2.20 exibe o método concreto do controlador, para receber a abstracção com os boletins de eVoto.

Um último apontamento: para permitir ajustar o tamanho (em *bytes*) aceite pelo IIS, dado o elevado número de eVotos – cada eleitor com múltiplos boletins, diversos órgãos e listas – teve-se de ajustar o tamanho do conteúdo do `POST`[16].

```
<configuration>
  <configSections>...</configSections>
  <appSettings>...</appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.7.1" />
    <httpRuntime targetFramework="4.7.1" maxQueryStringLength="8192"
     maxRequestLength="3500985" executionTimeout="1500"/>
  </system.web>
  <system.webServer>
    <handlers>...</handlers>
    <!-- Parte nova, limitação a 100 megas -->
    <security>
      <requestFiltering>
        <requestLimits maxAllowedContentLength="100000000"/>
      </requestFiltering>
    </security>
    <!-- fim parte nova -->
  </system.webServer>
  <runtime>...</runtime>
  <system.codedom>...</system.codedom>
  <connectionStrings>...</connectionStrings>
  <entityFramework>...</entityFramework>
</configuration>
```

Figura 2.21: WebAPI: Web.config.

2.6.3 Implementação do cliente

Como se indicou, facilmente qualquer plataforma será capaz de produzir um pedido HTTP. No âmbito do .NET, uma das formas de o obter será através da classe `HttpClient`. Esta incorpora mecanismos de tradução para JSON. Como tal a construção e colocação do pedido reduz-se a um pequeno conjunto de instruções, tal como exibe a figura 2.22. Do lado do cliente, dada a dimensão em milhares de boletins, tem-se de considerar o tempo de envio, do conjunto de todos e, após recepção, aguardar o processamento do serviço de contagem: cada boletim necessita ser processado (decifrado). Foram feitas estimativas de espera da resposta, com base num cenário de ligação até 1Mbps, sobre um conjunto de várias centenas de milhares de boletins, tendo-se determinado aceitável um valor máximo de espera (*timeout*) de 30 minutos. Na prática, o processo concreto de contagem da eleição para a OA, ficou pelos 3 minutos e 23 segundos. Ainda assim, bem acima do valor pré-definido de 100 segundos[17].

```
2 references | Ricardo Henriques, 362 days ago | 1 author, 1 change
private async Task<ResultadoWS> ChamaWSContagem(long idEleicao, string _baseAddress) {
    try {
        using (var client = new HttpClient()) {
            client.BaseAddress = new Uri(_baseAddress);
            client.DefaultRequestHeaders.Accept.Clear();
            client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue
             ("application/json"));
            client.Timeout = TimeSpan.FromMinutes(30);
            List<TuploVoto> postValues = (from _evoto_ in db.OA_Eleicao_votoE
                                          join _eleicao_ in db.OA_Eleicao_BoletinsVoto
                                          on
                                          _evoto_.idBoletim equals _eleicao_.idBoletim
                                          where _eleicao_.refEleicao == idEleicao
                                          select new TuploVoto {
                                              IdBoletim = _evoto_.idBoletim.ToString(),

                                              HashVoto = _evoto_.hashVotacao,
                                              IndicePK = _evoto_.indicePublicKey ?? 0
                                          }).ToList<TuploVoto>();
            HttpResponseMessage response = await client.PostAsJsonAsync("api/Contagem",
              postValues.ToArray())
                .ContinueWith(t => {
                    try {
                        return t.Result;
                    } catch (AggregateException ex) {
                        ex.Handle(inner => {
                            if (inner is HttpRequestException) {
                                System.Diagnostics.Debug.WriteLine("ws contagem - 2a");
```

Figura 2.22: Invocação de serviço de contagem.

Capítulo 3

Segurança & Invariantes

Se avistares um gigante observa a posição do sol,
e repara se o que vês não é a sombra de um anão.
– Novalis

A colocação deste capítulo, no fim, serve de metáfora para duas realidades: o descobrir que, por fim, não iríamos ter um consultor, alguém especialista em segurança que nos ajudasse nesta demanda; e, que a segurança não é vendida como prioritária e, portanto, é deixada para o fim...

A falta de tempo, o corte no orçamento e as mudanças nas equipas, de gestão e desenvolvimento, foram as forças que impulsionaram a transição do que poderia ser feito, para o que pôde ser feito, dentro do prazo de urgência e circunstâncias financeiras. Posto isto, o caminho adotado foi derivar ou construir uma solução, sempre que possível com base em bibliotecas existentes, com a ambição modesta de não construir um sistema impenetrável, mas antes tentar blindar as partes onde os componentes, ou onde as suas ligações estariam mais expostas. Como se deixou patente na introdução (capítulo 1), uma parte significativa das dificuldades assenta no facto do *sistema* – código, base de dados, comunicações – estarem ao dispor de quem tinha acesso, incluindo um núcleo de membros dirigentes da Ordem. Em resumo, aos aspectos descritos em 1.3.2, junta-se à preocupação de prevenir a intrusão externa, e a preocupação de impedir (pelo menos dificultar), a fraude interna.

A tabela 3.1 sintetiza a conciliação dos requisitos da CNE, para uma qualquer eleição, e os seis requisitos, enunciados por Bruce Schneier[11], para o voto electrónico. À esquerda apresenta-se o requisito a satisfazer, a condição, e à direita faz-se a ligação ao mecanismo adoptado.

Invariante	**Descrição**
A urna é legal.	3.1
Apenas eleitores autorizados podem votar.	3.2
Ninguém pode votar mais do que uma vez.	3.3
Ninguém consegue determinar o voto de outrem.	3.4
Ninguém consegue duplicar o voto de outrem.	3.5
Ninguém consegue alterar o voto de outrem, sem ser descoberto.	3.6
O voto de cada eleitor é levado a contagem e tabulação.	3.7

Tabela 3.1: Invariantes do voto (electrónico).

Vamos ver que estes problemas não nascem do voto ser electrónico, mas antes dele ser feito sobre uma rede, como se irá aprofundar. Desde já, em resumo pode-se afirmar que os problemas de segurança em redes (informáticas) se dividem em quatro áreas interconectadas: sigilo (*secrecy*), autenticação (*authentication*), não-repúdio (*nonrepudiation*) e integridade (*integrity control*).

Votar, mesmo no voto tradicional em papel, não é apenas o preenchimento de um conjunto de boletins que se colocam na urna. É, antes de mais, um protocolo. Um protocolo[11] *é uma série de passos, envolvendo dois ou mais intervenientes com o propósito de completar uma tarefa.* Tal significa que o protocolo estabelece uma sequência, uma ordem de acções, com um início e um fim. Envolve dois ou mais intervenientes. Se for apenas um interveniente a cumprir um conjunto de passos, não é um protocolo, mas uma realização ou actuação. O facto de haver um propósito significa que o protocolo pretende atingir um objectivo, um propósito.

3.1 A urna é legal?

O eleitor ao dirigir-se a uma urna (voto tradicional em papel), desloca-se fisicamente. Tal simplifica duas condições que no voto electrónico não estão asseguradas:

- A confiança na urna – ou seja através do reconhecimento fidedigno das pessoas que compõem a mesa eleitoral.
- Não haver intermediários – é o próprio que leva e coloca o voto na urna.

Vamos ver como se implementam no domínio digital estas estas abstrações. Recupere-se alguns conceitos quase esquecidos. Era comum sermos

confrontados com termos como:

- LAN (*Local Area Network*) - com extensão inferior a 10Km.
- MAN (*Metropolitan Area Network*) - entre 10km e 100Km.
- WAN (*Wide Area Network*) - acima dos 100Km.

Ainda hoje a extensão geográfica tem significado e relaciona-se com uma importante condicionante subjacente à arquitectura das redes, a qual se prende com a utilização de meios próprios ou dos operadores de telecomunicações. Numa rede de pequena dimensão a cablagem e o equipamento de transmissão (ou só este em redes wireless) são instalados e geridos pela instituição que a usa. Numa rede de grandes dimensões e geograficamente dispersa poucas organizações conseguem sustentar a infra-estrutura física de comunicação. Como tal, no caso do eVote, onde o eleitor podia votar a partir de qualquer lugar do mundo (com conectividade), a ligação entre si e a urna (num servidor *web*), seguiu a primeira das seguintes abordagens:

- A utilização (partilhada) da rede dos operadores telecomunicações;
- A implementação de uma rede privada, alugando aos operadores apenas os meios de transmissão.

3.1.1 Arquitectura física e lógica

A palavra rede no sentido acima descrito é um sistema de comunicação que permite a interligação de equipamentos informáticos heterogéneos, independentemente da distância a que se encontram. Contudo, para o utilizador final, a elevada complexidade destes sistemas está abstraída por diferentes níveis que vão desde sinais eléctricos (ou ópticos, ou de ondas rádio) nos meios de transmissão, até as interfaces de programação utilizáveis pelas aplicações. Seguindo a habitual estratégia dos sistemas computacionais, da divisão de um sistema complexo em diversos níveis de abstração, procurou-se criar um modelo que estabelece os principais níveis e o modo como estes se relacionam. Têm-se, desde de logo duas subdivisões, correspondendo à arquitectura física e à arquitectura lógica.

A arquitectura física de uma rede define os meios de transmissão, a topologia descrita pelos sistemas e as ligações entre os mesmos. A ligação entre nós da rede depende em grande medida da forma como é controlada a transmissão. Nas redes LAN a ligação entre nós é feita por um meio partilhado, onde a informação é enviada em difusão a todos os nós (*bus* por exemplo)

ou no estabelecimento de ligações ponto-a-ponto em que os sistemas se interligam aos pares (anel ou estrela). Num meio físico partilhado por vários nós tem de se estabelecer uma regra de controlar o direito de transmissão, ao passo que no segundo caso o controlo é mais fácil, mas o encaminhamento de dados entre nós não ligados directamente é mais complexo. Nas redes WAN, a grande escala impõe normalmente uma topologia ponto-a-ponto como a topologia malhada[51].

A arquitectura lógica procura dotar a rede de um conjunto de propriedades capazes de responderem ao endereçamento, desempenho, determinismo na entrega, tolerância a falhas, garantia de ordenação de pacotes, difusão, etc. O objectivo principal da divisão por níveis do modelo de comunicação das redes de dados é flexibilizar a adopção de uma arquitectura lógica mais adaptada ao campo de aplicação, independentemente da arquitectura física subjacente. O nível de separação entre a arquitectura física e lógica depende de rede para rede.

3.1.2 Protocolos *internet*: a utilização de *layers*

O modelo de referência OSI (*Open Systems Interconnect*) foi definido pela organização internacional de normalização *International Standards Organization* (ISO) como parte do esforço da obtenção de uma arquitectura de comunicações "aberta"[32].

O modelo estrutura a arquitectura de comunicações num conjunto de níveis ou camadas (*layers*) estanques, com funções e interfaces bem definidas. Cada nível possui uma interface, a qual define a funcionalidade que os serviços a esse nível disponibilizam à camada imediatamente superior, e um protocolo que define o formato e o significado das mensagens trocadas entre níveis correspondentes, nos sistemas que intercomunicam.

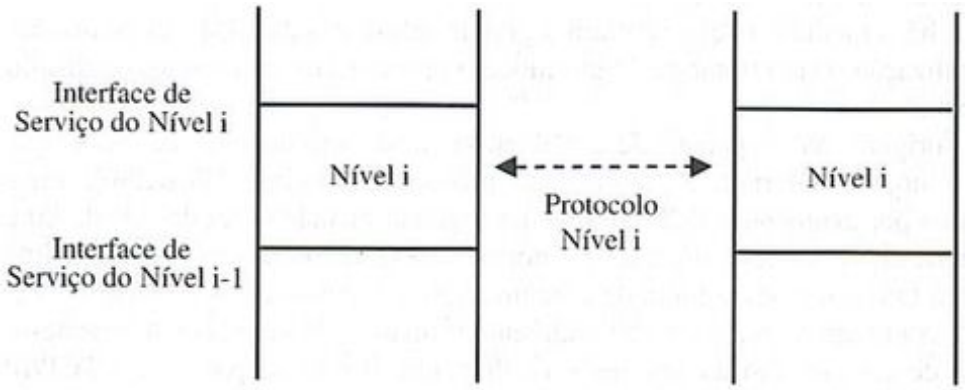

Figura 3.1: Interface entre níveis.

Cada nível é encapsulado no nível abaixo e inversamente, na recepção, é extraído o pacote que corresponde ao nível idêntico do emissor:

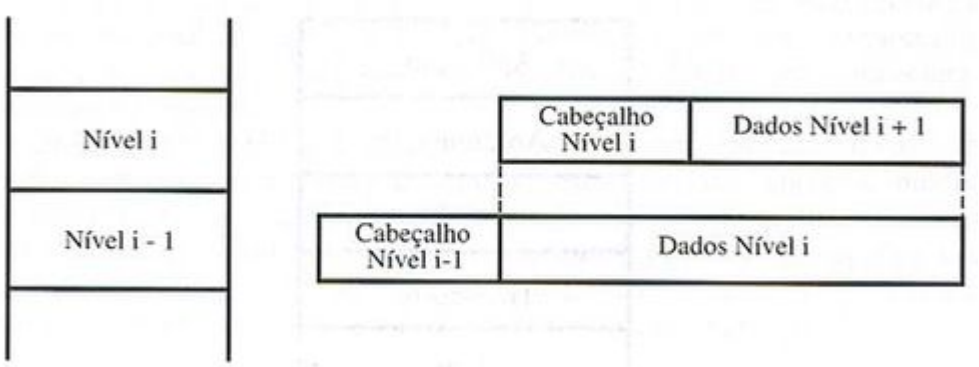

Figura 3.2: Encapsulamento entre níveis.

O modelo OSI estrutura-se em 7 camadas: física, lógica, rede, transporte, sessão, apresentação e aplicação. Na realidade esta abstracção nunca foi implementada com sucesso, e o sistema usado, comummente designado por TCP/IP, é uma aproximação deste, composto por 5 níveis (camadas ou *layers*). De acordo com a nomenclatura[1] usada por Tanenbaum [60] são:

- Aplicação – *application layer* - engloba os *application, presentation, session layer* do OSI.
- Transporte – *Transport layer*.
- Rede – *Internet layer*.
- Subrede – *Data link*.
- Meios de transmissão – *Physical layer*.

O eleitor preenche, através do seu *web browser* e a nível aplicacional, o conjunto de boletins a colocar na urna (*web application*), no servidor da OA. Ambas ferramentas estão no *application layer*. A esse nível dialogam no protocolo HTTP. Porém, o servidor eVote não aceita, para benefício do eleitor, HTTP, mas sim HTTPS. O *S* de segurança, resulta da adição do *Transport Layer Security* (TLS), sucessor do *Secure Sockets Layer* (SSL). Não se trata de um nível/camada nova do TCP/IP. Existe em termos lógicos ao nível do *presentation layer* do modelo OSI, ao serviço das aplicações.

[1] Em inglês.

O TLS tem neste processo dois aspectos fundamentais. O primeiro tem a ver com o título desta secção. A legitimidade da urna é obtida mediante uma função de verificação da autenticidade da urna (servidor eVote). Cada certificado TLS/SSL é validado por uma *Certificate Authority* (CA) fidedigna[31]; fá-lo para um servidor específico num domínio específico (num *website address*). O eleitor em papel aceita depositar o seu voto na urna porque, ora reconhece o local (junta de freguesia/escola), ora porque até conhece as pessoas da mesa de voto. Aqui, o eleitor deve munir-se do mesmo cuidado e validar que reconhece o endereço do *website* como o correcto, o qual deve ser publicado por outros canais.

O segundo papel do TLS entra logo de seguida em acção. O pedido do *browser* ao servidor – por exemplo para apresentação das opções do boletim – só é transmitida após ambos iniciarem uma sessão segura. Estabelecem para essa sessão uma chave, não repetível, para a comunicação iniciar-se. Cada sessão TLS/SSL consiste em duas chaves:

- Uma chave pública usada para cifrar (ocultar) a informação.
- Uma chave privada para decifrar (mostrar) a informação e repô-la na forma original, para ser lida.

O grau de segurança no eVote depende da qualidade destes intervenientes: *browser* e servidor.

A chave, no contexto aqui apresentado, é uma fórmula matemática, ou algoritmo, com o qual se cifra e/ou decifra a informação. Da mesma forma que uma fechadura de uma porta, na qual quanto mais combinações (mais lados e "dentinhos") tiver a chave/fechadura, mais difícil será de abrir sem a chave, também num sistema digital, quanto maior for em *bits* o tamanho da chave, mais forte o mecanismo de cifragem, para o mesmo algoritmo.

3.2 Apenas eleitores autorizados podem votar

A importância deste invariante é tal que existe desde há muito no voto tradicional, sob a designação comum de caderno eleitoral. No caso português, o Arquivo Nacional Torre do Tombo, indica ter dados parciais que remontam a 1926. Porém, há registos do final do séc. XVII, relativos aos seus predecessores, originalmente designados de *poll books*[63], no Reino Unido.

No caso do eVote, os eleitores eram compostos por membros da OA, portanto arquitectos, com a exigência de terem a sua situação regularizada. De acordo com o modelo de autenticação, em 2.4.4, a autorização de acesso ao sistema eVote baseava-se na validação do utilizador estar inscrito no caderno

eleitoral, do qual o eVote toma o perfil de *relying party*. O componente da segurança que foi necessário desenvolver foi o mecanismo de geração do PIN, para envio por SMS[2], como anteriormente descrito em 2.5. Havia duas facetas a acrescentar:

- Utilizar a boa prática de um canal alternativo e pessoal, através do envio de um código para o telemóvel do eleitor – desta forma, caso por descuido do eleitor, a sua conta estivesse comprometida, o usurpador ainda teria de ultrapassar este passo, para votar em sua vez.

- A outra parte a resolver era o perigo vindo de dentro – a um administrador de sistemas que tivesse forjado o acesso, apenas tinha agora, de espreitar a base de dados e usar o PIN enviado e lá registado.

Neste cenário, se o PIN a confrontar estivesse guardado na base de dados, e o elemento maligno fosse um administrador de sistemas, poderia obter o PIN e votar em lugar do legítimo membro. Como esconder de entre a tecnologia que havia disponível? Qualquer programador .NET facilmente encontra classes nativas capazes de implementar algoritmos, designados na literatura de *secure hash algorithms*. Mas também sabia, tal como qualquer programador com limitado conhecimento em segurança sabe que algoritmos de cifragem como o *Message-Digest Algorithm* (MD5) ou o *Secure Hash Algorithms* (SHA)1[56] oferecem uma proteção limitada. Como o .NET tem ao dispor SHA512 pareceu trivial utilizar este algoritmo, por ter uma reputação muito superior. Porém, se o objectivo era esconder de um administrador de sistemas mal intencionado, tal não serviria. O mesmo teria acesso ao conhecimento de que a *hash* resultante viria de uma mensagem original cujos limites eram números, 4 números no intervalo [0000..9999]. Ora, o SHA512 para a mesma entrada resulta sempre na mesma saída. Portanto, o administrador mal intencionado nem teria de construir uma pequena *rainbow table*[43], mas apenas um mapeamento da pré-computação das *hash* que resultam das combinações de 0 a 9999. Demoraria milésimos de segundo para mapear com o PIN original.

Por si, a computação do *hash* resulta num valor sempre constante para o mesmo PIN. O que a imagem 3.3 mostra, é que apenas a complexidade para reverter aumenta à medida que o número de *bits* de saída aumenta, de 128, 160, 256, 384 e 512, respectivamente, para os algoritmos MD5 e variantes do SHA.

[2]Ou email, se o primeiro método não estiver disponível.

	PIN = 1234
MD5	gdyb21LQTcIANtvYMT7QVQ==
SHA1	cRDtpNCeBiql5KOQsKVyrA0sAiA=
SHA256	A6xnQhbz4Vx2HuGl4lXwZ5U2I8iziLRFnhP5eNfIRvQ=
SHA384	UE8AjI/Piy7V383nUvxUZKuLoGQhXZxbX8SGrz2auMgbFHhRgNKtfO4at5KtRHmM
SHA512	1ARVn2Auq2/WAqx2gNrL+q3RNjAzXpUfCXrzkA6d4Xa22yhRLy4AC50E+6UTPoscbo31nbOoq51gvkuXzJ6B2w==
	PIN = 4321
MD5	2TWRvfeGDh5O4vynmZESFQ==
SHA1	1fEuU6GCwGK2vzDBRFFT+v8SJpo=
SHA256	/iWStCpyfpd/BVlHOFtwnMgrFrmof4jGq/OQDWXQzcM=
SHA384	eoDhyKjxRWrmwnNQh+vK9gvPbcF8xnfXyxZVFswRQ2LutD/2ZOLp+IUvn6t9YYeo
SHA512	fi/qyV3NfR34AzReGXNpr0sVbk56lfyylVvbuzoRr9i7nTWTG/FVETcLGBQ+OLAbkD9Vxey97Ur5mTRgL83zjA==

Figura 3.3: Exemplos do cálculo do *hash* de PINs.

Para circundar esta limitação crítica para a segurança, quando o ataque viesse de dentro, era necessário algo mais. Desde logo era preciso compreender o que de facto fazia que o resultado fosse sempre o mesmo. O tema da segurança foi das poucas áreas que durante o projecto foi, imprescindivelmente possível e necessário, aprofundar o conhecimento, através do estudo de bibliografia, quer por não ter experiência no campo, quer por não ter acesso ao tempo[3] de quem, o tendo, o pudesse partilhar.

3.2.1 *One-Time Signatures*

Os esquemas mais comuns de segurança como *Digital Signature Scheme* (DSA) ou SHA baseiam-se no princípio da dificuldade em resolver problemas logarítmicos discretos e no problema de factorizar grandes números[9].

O motivo do resultado da computação do *hash* manter-se constante, é implícito à sua própria natureza: porque são funções de *hash*. Uma função de *hash* tem como capacidade mais relevante conseguir distribuir uniformemente as chaves pelos vários índices, de forma a minimizar o número de colisões. Esta parte era conhecida apenas pelo percurso profissional anterior.

Uma *cryptographic secure hash*, é uma função de *hash*:

$$H : \{0,1\}^* \rightarrow \{0,1\}^s$$

Onde, a partir de um conjunto arbitrário de *bits* (valor ou texto original), digere (produz) uma resultado de s *bits*. No caso do MD5 $s = 128$ *bits*.

[3]Não é suficiente estar numa instituição que tem nas suas fileiras peritos na matéria para que esse conhecimento seja aplicado. Não existe osmose de conhecimento. É necessário orçamentar e pagar a tais recursos.

A função de *hash* será tanto mais criptograficamente segura quanto mais resistente ela for. Essa análise é, na literatura[7], decomposta em 3 categorias:

- *Preimage resistant* – significa que é tanto mais resistente quanto mais difícil for encontrar h, tal que $h = H(m)$.
- *Second preimage resistance* – tanto mais, quanto mais difícil for, conhecido m_1, encontrar um m_2 tal que $H(m_1) = H(m_2)$.
- *Collision resistance* – e, finalmente, tanto mais resistente quanto mais difícil for encontrar um par tal que $H(m_1) = H(m_2)$.

Como neste caso se referiu, por se conhecer o conjunto de partida, com um máximo de 9999 opções, era sempre fácil ao administrador de sistemas pré-calcular o *hash* e, trivialmente anular a *preimage resistance*.

A solução passou por continuar a utilizar uma *cryptographic secure hash*, mas para baralhar o número de opções, fez-se adicionar um conjunto de *salt-bytes*. O *sal* não é mais do que uma *string* aleatória que é concatenada com o PIN antes de ser operada pelo SHA. Agora o mesmo administrador de sistemas já teria de gerar uma *one-time signature* para cada possível combinação. Ciente que o *sal* não é uma panaceia, e que o aumento do número de *salt-bytes* apenas protege contra os habituais dicionários de *hash* pré-computadas. Ainda assim, ao complementar com um curto tempo de validade do PIN, minimiza-se o risco descrito.

A figura 3.4, na página seguinte, mostra para dois PINS exemplificativos, 1234 e 4321, o efeito dos *salt-bytes* nos *cryptographic secure hash*: fica bem evidente que o resultado é aleatoriamente diferente, como se requeria.

	PIN = 1234
MD5	CT2sKh5J37kEDcqRh46n7SWcZA==
	m8lCsy1xdleSBxhRo+vzAaDpgw==
	4fDQj5SGEe3AmLCdAK6CnNly1A==
	(...)
SHA1	gYhZuolEWdT4lC0XtK5Ima/iUc2hk5k=
	0j1xx3zOCbZbYW2H/erOn60ulUN3zJo=
	l6hsAVDS8FuEyNGXE3ZRmyvvS35H9w8=
	(...)
SHA256	VsMafnm6Z4g9PSvkkeUc7Q30Ui+mB//FmJ9L20rAB0o39PY=
	nQTf3hs1lyo8X4GlA0bEBvfDEJg7pc0fRFJlcjLPiT3N52M=
	qnCly+3H/9emL6mjfWr3rz/Dd5eU8+1US3lgJCpyURLquts=
	(...)
SHA384	nbqQ9/lt4Ss0k+/HBL0nCmAQdkKM3hZGtkhbbrXqHnsVGDYo/bgkdRwO6Acwe/02OdGh
	JYSkqlAdOeBJTXizPgqFi6j84DnOhkdoOD1tTCKVJ08Y7iO52/mYuK8QjP9OFNTXe7Kt
	3jFs+oUS/GZ7zO8HrihLn5x50v9NpBKZVTt91jqp4j3LmoM00BwYuYms1FQuln9urzNK
	(...)
SHA512	MkLAZVDa16kxA1qclOMQS2PJl3jsgkZW5c8D+xl8V1afY47CsqTLcfGhhzomS1+Un6sWl3vus5rzU99iZ/bQ/nLRfw==
	FjpmS0S0kCenF8+Crx4hpzAohF3T1s+f2LuQTeaEhdmfqO2WjHiCsVBUC34eSR2xKtDR/PcvkKJY8nmyynfcUjncSQ==
	v6V8aBgKULEg+fhzQUDw9QahGsOhZUkhkhcrwfwFD1JnTU03Zg022bA5EblasKdP8DYhyMm67pnkQTyNmcK2xqK9xA==
	(...)
	PIN = 4321
MD5	dBx9OQVpdQDRX0HbLEdc/lR3nA==
	9B8mlO5ruJWqQw6FAU6KBa5h4g==
	Tr+pp9khKRWAhZ4HfersTL/xCQ==
	(...)
SHA1	kaPMcPvebmMwj7bBZzrawt5oO8xZlMY=
	X/9G+xJm3vJZ5zIGbAjq96k0l9ri1NU=
	/HUyrnomPpJbcr0ZV0N28DOfNrFPGiE=
	(...)
SHA256	X9fv4XkHQR49BFk45O8wFi69SerGEvyLUhdn6tbvgMVV8Uk=
	OGz50Q6gCzpHGsPvqpMKZLzTNXOvKP+cWaa0rgJFl5YxKZl=
	RGUBCExgfatT2BlegXsniwUlbRxyvKT1XG8LbHagaFAK7Tw=
	(...)
SHA384	Rb4ttZT7oUqag/2qBy+9QbUMBPXF/x7riCruA7tGHQOm40cogtqYS3c242EMoKb9lsaj
	VHrk/pOtO+tplKeUwQt9cj3PZzG5itwVnh6sxpkKGlvzNYfz63Cu0xtyPbBQeALgQ/U4
	0+aM/O1Z/IJR9SHJF1BW6VMWmlM2tw29ieMmApwf5e+qfdOnlF2/r/6Ubccvz8uFZgN2
	(...)
SHA512	VUJbzwIU4RP+pKJpSsjwDE7DMA+NB+MvHdBX1RUcK7boamnwNX+Y7/OJdZptBvF0C0yJulpCNn+Ms1AWfk3LdYl7UA==
	SotI3W5XF7cfkmQYHAJtPWITrjdbAZ/NLpM5w833jVg5mCQ+oFiGjfafVAnrndA/T4bCfrwGm40LgmFuwe6/TSxt9w==
	LvGg3rlx+BLsvec9xiigUSHHLca6mbtRpsGnbHma7aTiWvqFSleIYrNy+m43ofAzhxlqtDtq6VifOeod2Zzqbbyby Q==
	(...)

Figura 3.4: Exemplos com 3 *salt-bytes*.

3.3 Ninguém pode votar mais do que uma vez

Este requisito é extremamente forte. Embora seja muito simples de enunciar é penoso de cumprir. O objecto do problema não é o eleitor querer novamente exercer o seu voto. Como a urna electrónica é única (centralizada), e o conjunto de valores a actualizar na base de dados, no momento do voto, são vistos e operados de forma atómica, tem-se que o proteger, neste sentido,

implementa-se num trivial `if-then-else`. Contudo se o administrador de sistemas quisesse podia repor o estado inicial, anterior ao voto. E com isso o eleitor podia votar novamente, ou contar como abstenção.

Assim, o máximo que se tentou no tempo disponível foi, quanto muito, garantir que *ninguém pode votar mais do que uma vez, sem que isso dificilmente não deixe rasto.* O princípio assenta na validação ou verificação do diário, ou *log* (jornal), do sistema de gestão da base de dados.

O diário contém a história das acções executadas pelo sistema de gestão de base de dados. Fisicamente é um ficheiro de registos armazenado num suporte estável (por oposição à noção de memória dinâmica ou volátil), e sobre o qual se assume ser capaz de sobreviver a falhas do sistema. A parte mais recente do diário é designada de *log tail* e é mantida na memória (volátil) sendo periodicamente forçada a passar para disco.

Para efeito de recuperação, cada página da base de dados contém um *Log Sequence Number* (LSN) que descreve a alteração a essa página. Um registo de diário é escrito sempre que ocorre uma das seguintes classes de acções:

- *Updating page* – após a modificação de uma página.
- *Commit* – quando uma transacção decide consignar, ela força à escrita de um registo no diário contendo o identificador da transacção. A transacção é assumida no instante em que o registo do diário é passado para a memória estável. Outros passos são dados, por exemplo, a remoção da transacção da tabela de transacções.
- *Abort* – quando uma transacção aborta; de seguida o *undo* é iniciado para esta transacção.
- *End* – quando uma transacção aborta ou consigna há um conjunto de passos a dar. Quando estas tarefas concluem é escrito um registo no diário com essa informação, nomeadamente com o identificador da transacção.
- *Undoing an update* – quando uma transacção é anulada, as alterações por si causadas são revertidas. Quando a acção descrita pelo *update record* é anulada (revertida), um *Compensation Log Record* (CLR) é escrito.

Figura 3.5: Exemplo de um *update record*.

Cada registo no diário contém: `prevLSN`, `transID` e `type`. O conjunto de todos os registos para uma determinada transacção é mantida sob a forma de uma lista ligada; tal permite recuar no tempo através do campo `prevLSN`: esta lista é alterada sempre que uma nova entrada é adicionada. O `transID` contém o identificador da transacção que causou a entrada no registo e o `type`. Outros campos podem existir, dependente do tipo de registo[45]. No sistema de gestão de base de dados SQLSERVER têm-se ao dispor uma relevante função `fn_dblog` capaz de devolver detalhes associados a cada operação. Neste caso, operações como `LOP_MODIFY_ROW` associada à tabela de registo do eVote seriam suspeitas, quando apenas se admitiriam operações `LOP_INSERT_ROWS`.

Current LSN	Operation	Transaction ID	Previous Page LSN	elements	Offset in Row	size
0000002a:00000280:0019	LOP_INSERT_ROWS	0000:00000566	0000002a:00000280:0015	3	0	0
0000002a:00000288:0004	LOP_MODIFY_ROW	0000:0000056a	0000002a:00000280:0019	6	19	16

Figura 3.6: Consulta ao diário

O exemplo da figura 3.6 acima, mostra um exemplo parcial num diário, correspondente à modificação de uma linha (operação `LOP_MODIFY_ROW`). Se uma situação destas estivesse presente no diário da base de dados, para a tabela de registo do voto electrónico, onde apenas deveria conter referências a inserções, deixaria rasto. A referência ao estado original, neste caso referido pelo *Previous Page LSN*, daria para repor o valor correcto. Se o administrador do sistema adulterasse e quisesse esconder os seus actos, teria dificuldade em o fazer, a menos que suspendesse o diário, ou o eliminasse no fim. Em qualquer destas situações, seria notado, mas era grave. Tal invalidaria o processo da eleição e obrigaria a repetir todo o processo eleitoral. Desconheço, não é da minha área de interesse, mas suspeito que é um ponto importante que não estará legislada para esta acção específica.

3.4 Ninguém consegue determinar o voto de outrem

No voto tradicional admite-se que o voto de cada eleitor não é determinado, pelo facto de assumirmos, que ninguém viu e que fica diluído no meio dos outros boletins. Para tal ser verdade, era necessário garantir que o boletim não estaria marcado e que os boletins não são segregados ao entrar na urna. Uma urna transparente ajudaria. Votos antecipados – tais como os exercidos de forma isolada numa autarquia – ou, por exemplo, a possibilidade do boletim de voto ter marcação química e invisível[21][1], subtraem confiança ao voto tradicional.

No voto electrónico, a parte mais óbvia e frágil do procedimento passa por associar a um eleitor o seu voto, mesmo quando se julga que o mesmo está protegido por uma cifra considerada segura. O armazenamento desta informação é necessária para responder ao invariante que será abordado em 3.7. De acordo com Huangyi[29], os sistemas de voto electrónico não apresentam um adequado nível de protecção. E os que apresentam apenas o fazem para o que designa de *privacidade imediata* – ou seja, são apenas capazes de proteger contra formas actuais de ataque à privacidade. Enquanto responsável pela implementação da cifragem e decifragem do eVote, partilho a opinião que o sistema desenvolvido cairá nesta classe, ao recorrer ao criptosistema de chave pública *Rivest-Shamir-Adleman* (RSA)[48] – e acessível pela biblioteca `System.Security.Cryptography` do .NET. Apesar de todo o cuidado, e de apenas ser guardada a chave privada num sistema independente (figura 2.8), não há garantias que novas técnicas e algoritmos de criptoanálise, recorrendo ou não a emergentes paradigmas de computação, não sejam capazes de abrir o voto sem recorrer à chave privada. Pode ajudar a possibilidade de, no fim do período de reclamação ou impugnação do acto eleitoral, a base de dados ser toda eliminada, se possível, garantir que o conjunto de máquinas usadas são formatadas. Mas tal vai depender do enquadramento legal, se o permite ou não, e no tempo que medeia o início do voto electrónico até ao encerramento de todo o processo. Ainda pode ser mais difícil, em cenários que recorram à colocação do sistema sobre modelos de *outsourcing*, cujas máquinas físicas não estão no controlo da comissão eleitoral.

A apresentação do método `DecifraBytes2String` tinha ficado prometido na secção 2.6. Aproveitando o contexto do mencionado RSA, coloca-se aqui um pouco de código.

```
public static string DecifraBytes2String(byte[] encryptedData, Parameters2Save p) {
    byte[] decryptedData = RSADecrypt(encryptedData, p.GetParameters, false);
    return RicRSA.RicConverteBytes2String(decryptedData);
}
```

Figura 3.7: Abertura do eVoto, método: *DecifraBytes2String*

A figura 3.7 mostra que recebe dois argumentos. O primeiro tem, como o próprio nome deixa perceber, o *hash* composto sobre o voto, à custa da chave pública. O segundo parâmetro, e com uma designação pouco criativa `p`, contém a parte correspondente à chave privada, que apenas o sistema de contagem conhece. Como, a par de cada eVoto é passado o índice usado, vemos chegar a este argumento os parâmetros que permitem, através da chave privada, recuperar a escolha do eleitor. `Parameters2Save` permite encapsular assim, um objecto da classe `RSAParameters` usado pelo sistema de decifragem, tal como se mostra na figura 3.8:

```
public RSAParameters GetParameters {
    get =>
        new RSAParameters {
            D = this.D,
            DP = this.DP,
            DQ = this.DQ,
            Exponent = this.Exponent,
            InverseQ = this.Inverse,
            Modulus = this.Modulus,
            P = this.P,
            Q = this.Q
        };
}
```

Figura 3.8: Parâmetros RSA

Para cifrar uma mensagem M o método RSA usa uma chave pública composta pelo par (e, n), ambos inteiros positivos. O valor de n é obtido de $n = p \cdot q$. Os números p e q são dois números primos mantidos secretos.

Entre outros passos, a biblioteca do .NET trata de colocar a mensagem original M, numa forma numérica, para permitir a aplicação da cifra. Criptografar a mensagem, passa por elevá-la ao enésimo módulo[4] da divisão da potência e por n, ou seja: $C \equiv M^e (mod\ n)$. A chave de decifragem é também um par de números positivos (d, n). Os primos p e q devem ser primos aleatórios muito grandes. Assim, embora o n seja público, os factores p e q

[4]Na teoria de números, dois números dizem-se congruentes em módulo, para um dado número, se ambos têm o mesmo resto da divisão por esse número.

ficam escondidos pela dificuldade de factorizar n. Ao mesmo tempo esconde a forma como d pode ser derivado a partir de e.

O d é escolhido sendo um número grande, aleatório, e primo relativo a $(p-1)\cdot(q-1)$. Tal significa que maior divisor comum (mdc), entre eles cumpre a condição de:

$$mdc(d,(p-1)\cdot(q-1))=1$$

Finalmente, o inteiro e e calculado de p, q e d, para ser o inverso multiplicativo, ou seja:

$$e\cdot d\equiv 1(mod(p-1)\cdot(q-1))$$

De regresso ao código, vê-se na listagem da figura 3.7 a invocação do seguinte método, o qual por sua vez invoca a biblioteca do .NET responsável pelo processo base de decifragem:

```
public static byte[] RSADecrypt(byte[] DataToDecrypt, RSAParameters RSAKeyInfo,
    bool DoOAEPPadding) {
    byte[] decryptedData;
    using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider()) {
        RSA.ImportParameters(RSAKeyInfo);
        decryptedData = RSA.Decrypt(DataToDecrypt, DoOAEPPadding);
    }
    return decryptedData;
}
```

Figura 3.9: Método: *RSADecrypt*

Provavelmente, se o código fosse hoje recompilado, passar-se-ia o argumento `DoOAEPPadding` a *true*, com base no conhecimento entretanto evoluído, e documentado por exemplo em [55].

Por fim, por simplicidade de manipulação, o valor do eVote em *bytes* é convertido numa *string*:

```
public static string RicConverteBytes2String(byte[] original) {
    UTF32Encoding ByteConverter = new UTF32Encoding();
    Decoder decoder = ByteConverter.GetDecoder();

    int tam = decoder.GetCharCount(original, 0, original.Length, true);
    char[] originalEmChars = new char[tam];
    decoder.GetChars(original, 0, original.Length, originalEmChars, 0, true);

    return new string(originalEmChars);
}
```

Figura 3.10: Método: *RicConverteBytes2String*

3.5 Ninguém consegue duplicar o voto de outrem

Comparativamente com os demais invariantes, será este o aspecto da segurança que poderá estar menos blindado, caso a fraude seja perpetrada ao nível do administrador de sistemas. O sistema foi montado com duas possibilidades. Um modo em que para acelerar o processo de voto e a disponibilidade do servidor eVote, a chave pública é escolhida aleatoriamente para ser usada, mas pode casuisticamente a aleatoriedade conduzir a sua reutilização; e noutro modo em que uma chave pública ao ser escolhida para cifrar um voto, fica marcada e sai do conjunto de chaves disponíveis. Se esta segunda modalidade estiver activa, a duplicação do voto, sem deixar rasto, é virtualmente impossível. Caso esteja apenas em uso a primeira modalidade, o administrador de sistemas só teria de copiar os dados de um eleitor para outro. Aqui haveria a necessidade de analisar mais uma vez o diário, tal como se referiu na secção 3.3, caso se tratasse de reescrever o voto de um eleitor com a cópia de outro, ou cruzar com o jornal (*log*) de outros sistemas (além do da base de dados), para conferir se a cópia de voto iria para um eleitor abstencionista.

3.6 Ninguém consegue alterar o voto de outrem, sem ser descoberto

A alteração do voto de outrem é, em tese, uma situação isomorfa a uma das situações já analisada. Tal acção é capturada no diário do sistema de gestão da base de dados, a qual regista sob a operação `LOP_MODIFY_ROW`. Como tal as mesmas considerações deste invariante resumem-se ao que foi descrito na secção 3.3.

3.7 O voto de cada eleitor é levado a contagem e tabulação

O último dos invariantes entrelaça-se com o requisito de verificação individual, com o qual se deseja validar, se o voto de um eleitor foi incluído no resultado global final, e ainda, com o requisito de verificação universal, no qual os observadores validam se o resultado da eleição corresponde aos votos expressos.

O processo de contagem é processado no fim do período eleitoral, por recurso a um *webservice REST* externo, descrito na secção 2.6.

A figura 3.11 exibe a vista parcelar do jornal, em resultado do pedido de contagem e devolução dos resultados eleitorais. Através do cruzamento de múltiplos indicadores, é possível determinar se o voto de cada eleitor foi contabilizado e, no final tabulado. Os principais indicadores são:

- O conhecimento dos eleitores presentes e válidos no caderno eleitoral e assinalados como tendo votado.
- O conhecimento dos eleitores cujo voto cifrado está presente na tabela de voto electrónico.
- O conhecimento do resultado do serviço de contagem de votos electrónicos.

Estes indicadores são apresentados aos membros da comissão de eleitoral, para ajudar a decidir quanto à validade das eleições.

```
==== inicio ====
Tam aproximado = 9539836 bytes, 9316 KB, 9.1 MB
20:05:13
Submeter 54824 votos...ok (OK)
Codigo http: OK
Mensagem: OK
Boletim = 212, Opcao = 21, Contagem = 1493
Boletim = 212, Opcao = 22, Contagem = 1933
Boletim = 212, Opcao = 23, Contagem = 2710
Boletim = 212, Opcao = 24, Contagem = 570
Boletim = 212, Opcao = VotoBranco, Contagem = 147
Boletim = 213, Opcao = 21, Contagem = 64
Boletim = 213, Opcao = 22, Contagem = 47
Boletim = 213, Opcao = 23, Contagem = 75
Boletim = 213, Opcao = 24, Contagem = 4
Boletim = 213, Opcao = VotoBranco, Contagem = 1
Boletim = 214, Opcao = 21, Contagem = 94
Boletim = 214, Opcao = 22, Contagem = 46
Boletim = 214, Opcao = 23, Contagem = 58
Boletim = 214, Opcao = 24, Contagem = 5
Boletim = 214, Opcao = VotoBranco, Contagem = 5
                          (...)
Boletim = 242, Opcao = 21, Contagem = 447
Boletim = 242, Opcao = 22, Contagem = 716
Boletim = 242, Opcao = 23, Contagem = 881
Boletim = 242, Opcao = 24, Contagem = 182
Boletim = 242, Opcao = VotoBranco, Contagem = 66
Boletim = 243, Opcao = 21, Contagem = 396
Boletim = 243, Opcao = 22, Contagem = 661
Boletim = 243, Opcao = 23, Contagem = 787
Boletim = 243, Opcao = 24, Contagem = 159
Boletim = 243, Opcao = VotoBranco, Contagem = 289
 ====== fim =====
== fim: 20:08:36
```

Figura 3.11: Contagem votos

Capítulo 4

Discussão e Resultados

A alegria está na luta, na tentativa, no sofrimento envolvido. Não na vitória propriamente dita.
– Mahatma Gandhi

4.1 Sobre a metodologia

Como antes se indicou, a integração do código era feita no mínimo uma vez por dia. Ainda assim, observam-se grandes interregnos no desenvolvimento, derivado à não alocação exclusiva ao projecto. Para além de ser uma prática desaconselhada pela generalidade das metodologias de desenvolvimento, as trocas de contexto retiram *momentum* o qual se traduz em perdas de produtividade, no projecto em causa e nos outros que se intercalam. Estimo um impacto mínimo com valores no intervalo entre 20% a 30%.

Figura 4.1: GitLab: vista de Abril 2019 a Abril 2020.

Adler e Benbunan-Fich[5], indicam que ocorre multitarefa quando, um indivíduo altera o foco da atenção para executar várias tarefas independentes, de forma concorrente. A noção de independência sugere que as tarefas são

autónomas. A noção de concorrência implica que estas múltiplas tarefas são executadas com algum tipo de sobreposição temporal.

O estudo destes investigadores reveste-se de particular relevância por centrar-se na interacção homem-máquina, *human-computer interaction* (HCI). Mais, o estudo foi desenhado para focar-se nas consequências de executar múltiplas tarefas, não correlacionadas, num mesmo dispositivo tecnológico, ou seja, excluindo todas as outras acções como comer ou ver televisão, enquanto usam o computador.

A resolução de várias tarefas pode reduzir-se a três estratégias de execução:

- Sequencial - onde a tarefa seguinte começa após a completação da anterior, sendo o grau zero de concorrência: não será o que interessa aprofundar, dado que a organização de actividades profissionais dos elementos de equipas de desenvolvimento, virtualmente não consegue obter esta configuração, salvo em períodos muito curtos.

- Paralela - onde todas as tarefas são atendidas em simultâneo e, como tal, existe o máximo grau de concorrência: como atenção humana está limitada a apenas conseguir verdadeiramente o paralelismo, quando os tipos de atenção não se sobrepõem, como é o exemplo clássico de escutar música enquanto se escreve, ou conduz e, por isso não haverá também interesse em aprofundar.

- Intercalada - onde as tarefas são desenvolvidas com uma observável troca de contextos: quando ocorre mudança de atenção ao nível cognitivo, e portanto, parece que as tarefas são executadas em paralelo, quando de facto são à custa de trocas de focos de atenção.

Há mais de um século foram evidenciadas as relações curvilíneas entre estimulação (*arousal*) e desempenho (*performance*). Em [66], Yerkes e Dodson demonstraram a existência de um ponto óptimo de estimulação que induz melhorias no desempenho. Contudo, baixo ou níveis elevados de estimulação trazem o desempenho para um subnível óptimo.

Diferentes exigências de trabalho provocam em cada indivíduo variações de estímulos. O incremento na carga laboral convoca uma maior estimulação com vista à maior mobilização de recursos cognitivos. Por isso, alguns autores defendem que abaixo de um nível de estimulação, o indivíduo perde motivação em se manter focado na tarefa em curso. Significa pois, que uma segunda tarefa, ao aumentar o estímulo, possa melhorar o desempenho da primeira tarefa. No outro extremo, sob elevados graus de estímulo, acaba por não conseguir lidar com as exigências da situação e, invariavelmente, o desempenho irá baixar.

Trabalhar em várias tarefas em simultâneo, ou ter múltiplos objectivos em mente, aumenta o grau de estimulação e implica conseguir manter o estado de informação, associado a cada tarefa. Para a memória, mudar uma tarefa implica adiar um objectivo, o qual para ser retomado, requer relembrar a informação associada da actividade suspensa.

O que o estudo indica: os indivíduos que se empenham em múltiplas tarefas irão ter um desempenho melhor do que aqueles que apenas se dedicam a uma só. Mas irá surgir um ponto de decremento do desempenho, onde a partir do qual, mais intercalação irá significar uma acentuada perda de desempenho. Do ponto de vista de programação, e mesmo dentro de um mesmo projecto, pode levar a um nível já considerável de tarefas a serem desenvolvidas em concorrência. Contudo, como o contexto é o mesmo mantém-se gerível. Substancialmente distinto é quanto a troca de contexto é operada por concorrência entre projectos, de propósitos distintos, com distintos *stakeholders*, e por vezes ainda, linguagens de programação distintas. Tal força um forte consumo de recursos cognitivos. É normal um programador necessitar de recuperar para a sua memória, dezenas de classes e dezenas ou mais de métodos[1] para regressar ao projecto intercalado. Bem distinto em termos de desgaste cognitivo, quando comparado com a troca de contexto para enviar um email, ou actualizar uns indicadores no *excel* de outro projecto.

Outros dois pontos importantes do trabalho destes autores são:

1. A evidência de que, na presença de uma tarefa exigente, a mesma funciona como dissuasora da alternância.

2. A relação entre multitarefas e desempenho depende da métrica para avaliar o desempenho.

    - Se o desempenho for medido enquanto produtividade (ie. eficiência de desempenho), aqueles com níveis médios de multitarefa terão melhor produtividade do que aqueles com baixo, ou muito alto índice de multitarefa: uma espécie de U invertido.
    - Se o desempenho for avaliado enquanto precisão de resultados (ie. eficácia de desempenho) a relação corresponde a uma curva descendente, onde a concretização de mais tarefas, em concorrência, conduz a uma acentuada perda de precisão – perda de qualidade.

Por fim concluem: a multitarefa como um fenómeno contemporâneo em linha com o ritmo de vida acelerado, num mundo conduzido pela tecnologia. Em casa, no trabalho, na escola, o indivíduo já não se foca numa actividade

[1] Num total de centenas ou milhares de linhas.

de cada vez, e tende a coreografar múltiplas tarefas em simultâneo. Embora a vantagem da multitarefa possa ser a ilusão da produtividade, a desvantagem é que tem potenciais efeitos negativos no desempenho.

4.2 Sobre boas práticas

A automatização de testes unitários, foi usada, como se referiu na secção 2.2. E tal como na maioria dos IDE's (*integrated development environment*), o Visual Studio, possui ferramentas ou extensões, que auxiliam o programador na criação e gestão de testes unitários.

Os testes de integração devem ser eles também uma forma de testes automáticos[42]. Os testes de integração devem ser desenhados para testar múltiplos pontos de integração do sistema: devem assegurar que as API's com outros módulos, os formatos de dados e as *interfaces* programáticas, com outras classes, desenvolvidas pela equipa, estão acoplar à medida que o código da solução é desenvolvido.

Em [37], Lacey apresenta-nos a importância do aumento de produtividade através da utilização de testes automatizados de integração e aceitação, figura 4.2.

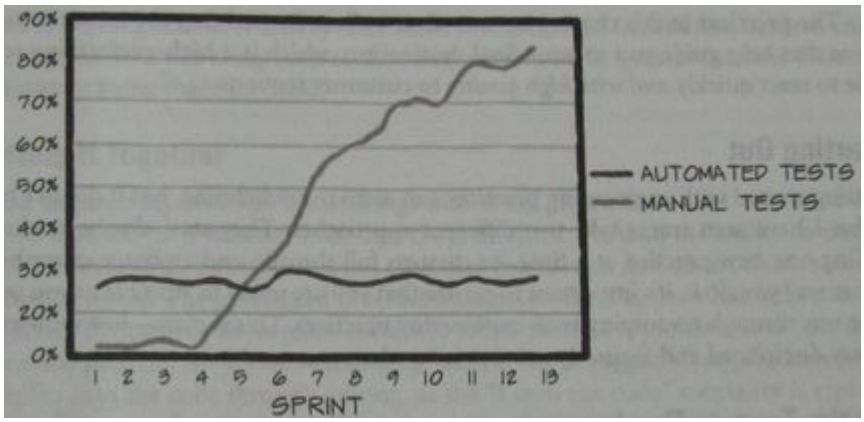

Figura 4.2: Testes de integração e aceitação: automatizada vs. manual.

Os testes de aceitação são desenhados para emular o comportamento do utilizador, como era o caso em mãos, quer com as fichas de inquérito, quer com o eVote. É possível testar a UI (*user interface*) através da escrita de código compilado ou *scripts* que simulam o comportamento do utilizador – incluindo os seus erros. Ferramentas como *Coded UI* da *Microsoft* podiam ser alternativas para tal desenvolvimento. Apesar da *Microsoft* reportar o *Coded UI* como *deprecated*, cresceram alternativas *open source* tais como

Selenium[25] e *Appium*[44], para testar quer ambientes *Web* quer *desktop*[2]. Tal não foi feito. Não foi feito porque muitos gestores de projecto, sofrem de desactualização, muitas vezes desconhecem que tal é possível, ou não têm formação para calcular em que momento passa a ser mais produtivo reservar um espaço ou um perfil para definir e construir tais testes, quer sejam de aceitação, quer sejam de integração. Números! É através de números que se justifica perante o cliente final a necessidade de considerar, no âmbito do projecto, determinadas opções.

Os estudos indicam, que apesar de um esforço adicional, a automatização de testes de aceitação e de integração são um importante contributo para garantir que a equipa desenvolve bem. Primeiro, facilita a equipa a ter um contínuo ciclo de *feedback*[3], a outro nível, mas semelhante à integração contínua e testes unitários. Depois, porque é de todo importante evitar longas fases de teste no fim de cada *sprint* ou *release*.

Segundo, é mais barato corrigir erros quando eles ocorrem (no caso da informática, quando são codificados), do que passado um conjunto de *sprints* ou pior, quando descobertos pelo cliente final.

Terceiro, os testes de integração automáticos ajudam a perceber, quando resultarem problemas após a adição de ligações a outros sistemas, se os mesmos têm origem interna (no código da equipa) ou vêm do sistema externo.

Mais uma vez se reforça, que toda a equipa (desenvolvimento, gestão e clientes) devem estar cientes desta abordagem, uma vez que os testes de aceitação devem ser escritos antes do início da *sprint*. Dessa forma quando o gestor do projecto (*manager* do XP, ou *product owner* do Scrum) entram na reunião de planeamento, a equipa sabe o que necessita de provar ao cliente do ponto de vista da *interface* final. Quando não se investe na automação, a cada *sprint*, o débito aumenta: no tempo necessário para a execução de testes manuais, drenando importantes recursos à equipa, como documenta a figura 4.2.

4.3 Sobre o *Entity Framework*

Sem por em causa o mérito e a utilidade desta *framework*, não poderia deixar de ser honesto com o leitor e confessar que de início tive algumas reticências na sua utilização. Assim, logo nos primeiros passos do projecto, decidi efectuar um conjunto de testes de desempenho comparativos, sobre uma outra

[2]WPF, *Winforms* e Win32.

[3]E tanto melhor se todo o projecto alinhar com uma alternativa ágil, seja, XP, Scrum, ou outra.

base de dados, uma vez que a base de dados de inquéritos e eVote não estava sequer desenhada.

Os resultados que pretendo partilhar foram inicialmente feitos sobre o EF v.5 tendo agora, para a escrita deste documento, feito a sua reescrita (recompilação) para a v.6, sendo sobre esta versão que se apresentam os resultados e compostos os grafismos de comparação. Para termo de comparação foram escritas duas simples *console applications*: uma com a utilização do ADO.NET (*connected*), com uso de *DataReaders*[20]; outra com LINQ e *DbSets* sobre a EF v6.

A primeira operação utiliza uma simples tabela com uma amostra de 40 registos e 6 atributos. Devolve todos os registos de acordo com a chave primária, projectando 2 dos 6 atributos (campos) da tabela.

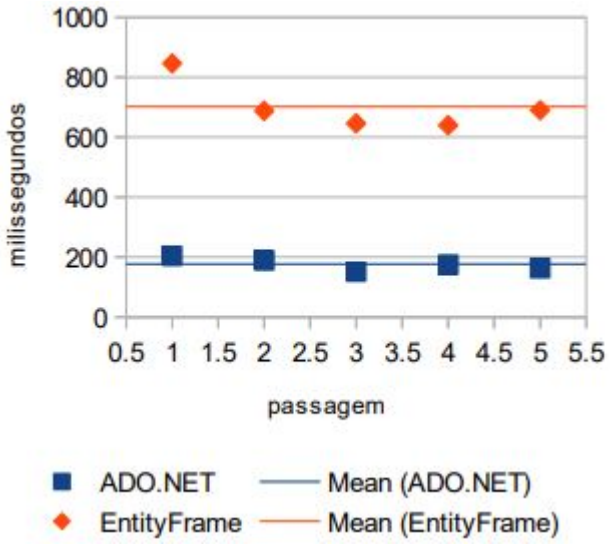

Figura 4.3: EF vs. ADO.NET: operação 1.

Em ambos os casos, percebe-se que a primeira execução trouxe um tempo superior às demais execuções da mesma operação – pela necessária passagem do código fonte compilado em metadados e em MSIL (*Microsoft Intermediate Language*) para o *assembly* concreto da máquina destino. Nesta operação o desempenho do ADO.NET é quase 4 vezes melhor do que da EF. Em média o tempo de execução do ADO.NET foi de 176.4 milissegundos e da EF de 702.0 milissegundos.

A segunda operação incide sobre a mesma tabela, os mesmos registos, mas pede que a sejam devolvidos os dados ordenados por ordem alfabética sobre um campo *varchar* não indexado.

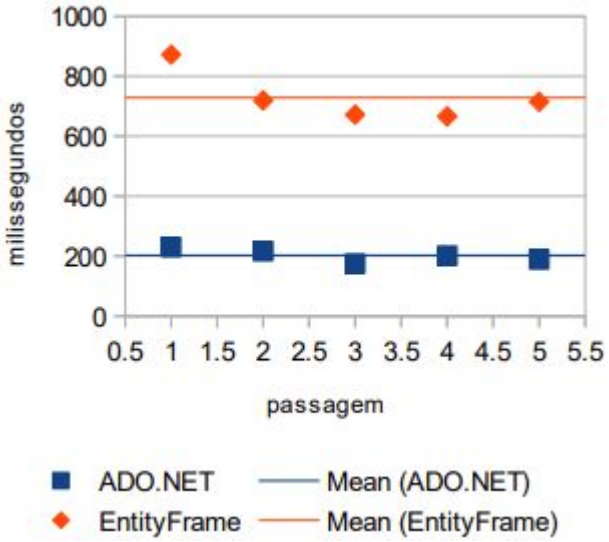

Figura 4.4: EF vs. ADO.NET: operação 2.

Em ambos os casos o desempenho piora, ficando novamente o ADO.NET melhor classificado. Contudo, o rácio de agravamento do desempenho, causado pela ordenação, subiu em 15% no ADO.NET contra 4% na EF.

A terceira operação requer a junção (*inner join*) entre uma chave estrangeira entre mesma tabela de 40 registos com outra de apenas 28 registos, sendo feita uma selecção dos registos que respeitam uma simples condição (*maior do que*), sobre um campo da tabela de junção.

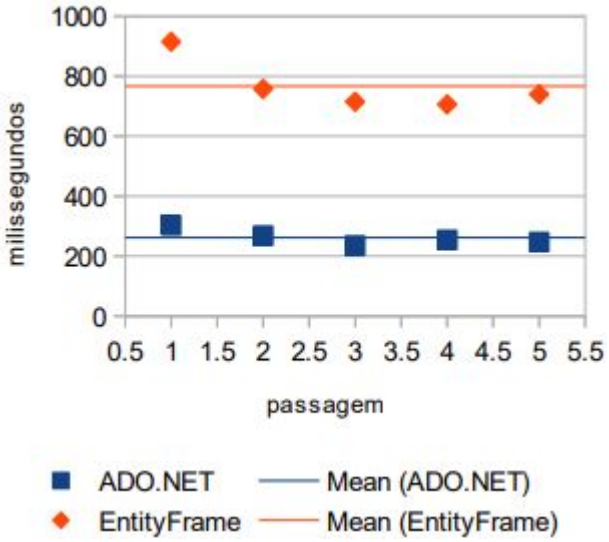

Figura 4.5: EF vs. ADO.NET: operação 3.

Sem surpresa o ADO.NET fica melhor classificado. Contudo nesta operação a diferença de desempenho cai pela primeira vez para valores inferiores a 3, em relação ao desempenho da EF, em concreto o desempenho médio do ADO.NET foi 2.93 melhor que a média do desempenho da EF.

Era ainda importante validar se a EF seria mais eficiente no contexto de uma operação de edição (*update*). Assim, a experiência incidiu no incremento de um campo *integer*, não chave, da tabela de 40 registos, e com uma condição associada ao mesmo campo a incrementar, ou seja, o incremento era condicional.

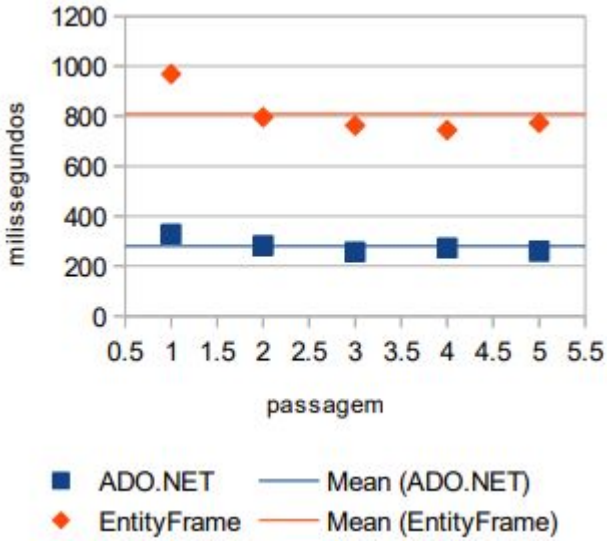

Figura 4.6: EF vs. ADO.NET: operação 4.

Como se observa, o ADO.NET foi novamente melhor, tendo em média feito a actualização dos registos em 279.8 milissegundos, enquanto a EF precisou de 809.2 milissegundos.

É claro que a EF ao permitir um grau de abstracção superior deixa uma maior pegada em termos de memória e de desempenho. Mas, esse esforço poderia ficar diluído se a amostra fosse superior. Por isso, ainda foi feito um outro teste. Desde logo, utilizou-se uma base de dados diferente – com mais registos. Além disso, quis-se, propositadamente, sobrecarregar do lado do cliente, com o intuito de analisar se, neste caso, a EF representava uma vantagem. Assim, sobre uma tabela com 11648 registos filtram cerca de 2.4% dos registos, por selecção sobre um campo de texto, não indexado. De seguida, em ciclo, faz-se a actualização noutra tabela, de um campo de síntese, com base nos valores, de cada registo seleccionado.

Como se observa, manteve-se o padrão de melhor desempenho para o

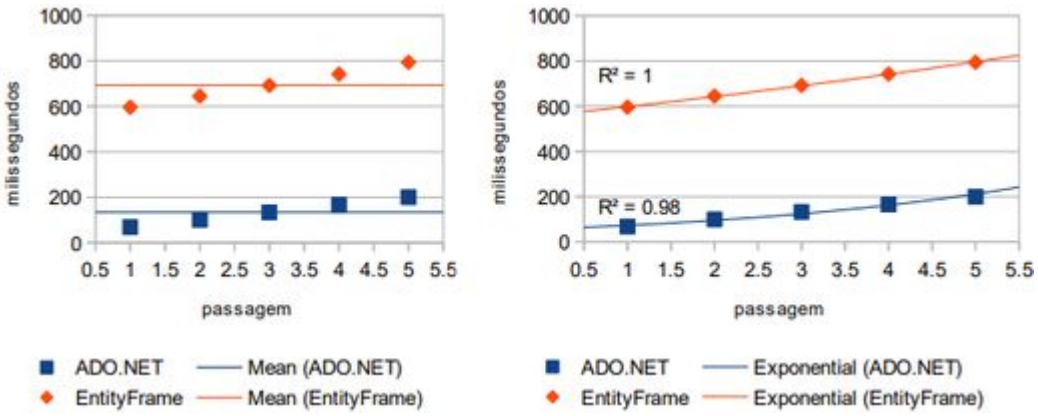

Figura 4.7: EF vs. ADO.NET: operação 5 – desempenho e crescimento.

acesso mais baixo nível, via ADO.NET. Pode ainda ler-se, pela figura 4.7, que a execução em ciclo é penalizada a cada iteração, com um crescimento linear R^2=1, no caso da EF, enquanto que o ADO.NET obtém um valor melhor de R^2=0.98).

Finalmente e para dificultar ainda mais o desempenho, fez-se uma pequena alteração ao código, de forma a que a cada iteração forçasse à sincronização no permanente. Os resultados correspondem à figura 4.8.

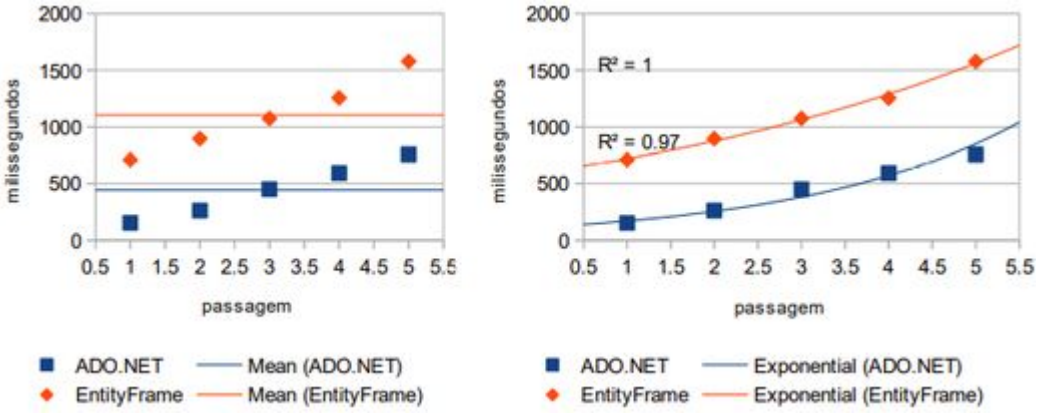

Figura 4.8: EF vs. ADO.NET: operação 6 – desempenho e crescimento.

Em resumo, a equipa de desenvolvimento deverá avaliar o compromisso entre facilidade de desenvolvimento, navegação na estrutura de dados e o desempenho. No sistema desenvolvido, para a equipa que tínhamos, dada a pequena dimensão da população de eleitores e as urnas virtuais manterem-se

abertas por um período longo[4], não trazia necessidade questionar o uso da EF. Se o mesmo sistema fosse pensado para umas eleições legislativas de âmbito nacional, dado o superior número de potenciais eleitores, concentrados num conjunto de algumas horas, imporia pensar em utilizar o máximo de soluções que privilegiassem o desempenho.

4.4 A votação: processo e resultados

Por não ter participado nas primeiras reuniões, até à saída do colega em doutoramento, não consigo confirmar de fonte independente, se foi ponderada a possibilidade de, utilizar uma solução preconcebida como o Helios[3], em lugar de definir de raiz o sistema electrónico de votação, eVote, ou quais os motivos [4] que conduziram à opção escolhida.

O processo de votação para a OA decorreu, entre as 12h00 do dia 17 e as 20h00 do dia 26 de Junho de 2020. Do total de 25454 potenciais eleitores, 19778 tinham a sua situação regularizada. Destes 6952 votaram, tendo a votação sido distribuída, maioritariamente com 6853 eleitores a utilizarem o voto electrónico e apenas 99 optaram por voto por carta.

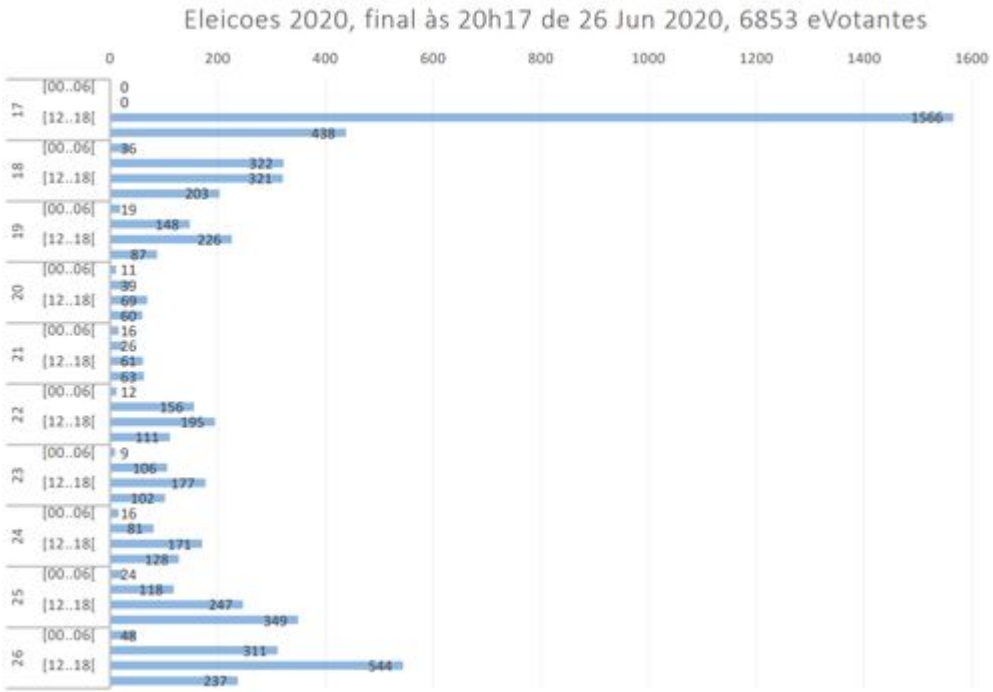

Figura 4.9: Distribuição de votantes eVote.

[4]Uma semana para uma população de cerca de 26 mil arquitectos inscritos.

A figura 4.9 mostra uma distribuição com dois momentos mais intensos, correspondentes ao período das 12h00 às 18h00, do primeiro e último dia de votação. Os dias 20 e 21 corresponderam ao fim de semana, período no qual o número de votos foi mais reduzido.

A figura 4.10 transmite outra perspectiva interessante. Apesar do reduzido número de votos por correspondência, diluída no universo do total de eleitores que exerceram o voto, o padrão de escolhas coincide com as preferências dos demais eleitores, que utilizaram o voto electrónico.

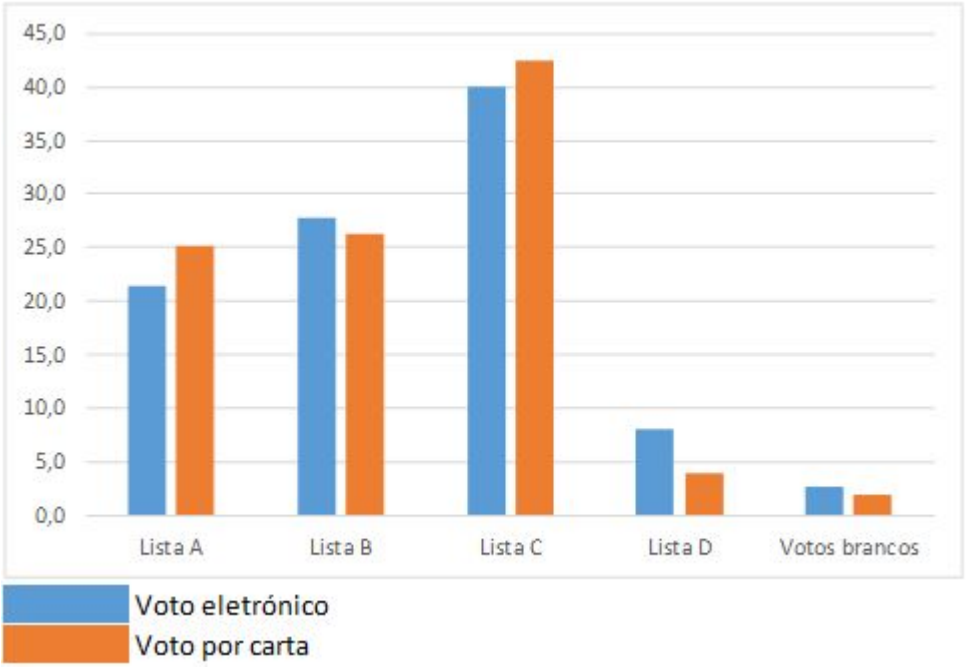

Figura 4.10: Distribuição percentual (orgão nacional).

A equipa de desenvolvimento e a OA fizeram vários ensaios, para permitir que os elementos de secretariado e os membros da comissão eleitoral pudessem, dar *feedback* por um lado, e ganharem competências e dissiparem todas as dúvidas de utilização. Na análise a esses ensaios, a equipa de desenvolvimento antecipou um comportamento sinistro dos eleitores. Chegados ao último boletim esqueciam-se de votar – ou seja votavam em branco – com a ânsia de submeter e terminar a votação. Como tal, a equipa de desenvolvimento adicionou uma caixa de confirmação para que o eleitor se apercebesse que surgiu um novo botão que não o de passar ao boletim seguinte, mas antes que se tratava de submeter as escolhas do eleitor. Não obstante, ao analisar novamente os resultados, voltou-se a detectar o mesmo desvio percentual no

número de votos brancos no último boletim. Com o pequeno conhecimento acumulado deste projecto, parece importante transmitir a necessidade de reforçar a formação, mesmo numa comunidade com elevado grau de literacia e conhecimento. O voto electrónico não é como o voto em papel.

4.5 Outras notas

Permita o leitor deixar duas observações gerais, sobre dois pontos, cujo impacto concreto no desenvolvimento deste trabalho não consigo quantificar.

4.5.1 Os *open-spaces*

Aproveito para deixar um pequeno apontamento sobre o desempenho humano e a sua relação com os outros. Sempre me questionei, porque mesmo em perfeito silêncio (ou quase) o meu desempenho criativo parecia ficar facilitado na ausência de outras pessoas no mesmo espaço. Muitas vezes procurava chegar mais cedo, quando o espaço estava vazio, e acabei por deduzir, erradamente, que tal se devia ao cérebro, de manhã cedo, *estar mais fresco*. Porém parece-me que, não só estarei errado – tanto é que por vezes ao fim do dia tinha desempenhos de igual qualidade, como não acontece só comigo. Está documentado, como por exemplo em trabalhos da NASA, nos quais apresentam a quantificação de múltiplos factores com efeito na qualidade e desempenho. A parte que me refiro, e que não imaginava sequer possível, é o quanto pode afectar a presença de outros, no mesmo espaço.

O desempenho, segundo o estudo em [12], é facilitado em actividades simples ou bem interiorizadas, quando em presença de outros. Porém o desempenho sofre degradação em tarefas novas (portanto mal interiorizadas), ou em tarefas complexas. Os processos de investigação e desenvolvimento, não assentam em tarefas simples ou repetíveis. Como tal, a disposição dos investigadores em *open spaces*[5] será, no mínimo, questionável – nomeadamente em processos (momentos) de criatividade ou na execução de tarefas complexas.

4.5.2 O *acordo* ortográfico

O motivo pelo qual tentei manter a escrita no português, da revisão de 1945 passa, não por motivos nacionalistas ou de índole mais ou menos política, mas apenas por me ser mais natural, dado o contexto temporal em que fiz a escola primária. Fui tentar perceber se os artigos n.º 1 e 2, do Decreto do

[5] Que nem era o meu caso, um pequeno gabinete para três almas.

Presidente da República n.º52/2008, enquadravam este documento nos casos obrigatórios indicados. Como desconfiava, fiquei sem perceber. Claramente como indivíduo não caio na obrigatoriedade indicada, e passo a citar *"(...) ortografia constante de actos, normas, orientações ou documentos provenientes de entidades públicas, de bens culturais, bem como de manuais escolares (...)"*. Depois o texto continua, e para perceber se força a esse enquadramento, fui consultar o regulamento de atribuição do título de especialista, nomeadamente, no seu artigo n.º 4. Este documento instancia o que lá se descreve como *"(...) trabalho original, de natureza profissional (...)"*, indicando ainda: *"O trabalho (...) não poderá ser de natureza académica ou científica (...)"*. Sob esta descrição julgo que também não cairá sobre o que descreve o referido decreto presidencial, o qual continua, e cito *"(...) e outros recursos didáctico-pedagógicos, com valor oficial ou legalmente sujeitos a reconhecimento ou certificação, à data existentes."*.

Decidi, portanto, manter a escrita no português de 1945, ficando ao dispor, para quem a título individual, não se importe de aguardar que componha uma versão, no português aceite pelos excelentíssimos Primeiro-Ministro e Presidente da República, então titulares, José Sócrates Carvalho Pinto de Sousa e Aníbal Cavaco Silva, respectivamente.

Capítulo 5

Conclusões

A ciência é a procura da verdade; não é um jogo no qual uma pessoa tenta abater os seus oponentes.
– Linus Carl Pauling

O contexto do projecto, com as componentes de inquéritos e do sistema electrónico de votação, denota a vontade da sociedade utilizar os sistemas de informação para aproximar pessoas, os seus membros, poder auscultá-los, e tornar um acto solene e privilegiado – como o poder votar – num acto simples, cómodo e acessível. A OA espelha bem, como em geral no mundo actual, as ferramentas informáticas tornam as distâncias mais curtas. Ao mesmo tempo evidencia os paradoxos entre a persistência numa antiga forma de ver a realidade, com fronteiras, secções, zonas, quando um número crescente dos seus membros estão dispersos, a trabalhar pelos quatro continentes, e onde a proximidade é feita pela partilha de valores, de vontades e de interesses, e não se estrutura em vizinhanças geográficas.

O segundo raciocínio que gostaria de destacar, fica bem evidente se aplicado noutro contexto, antes de trazê-lo de volta ao mundo dos sistemas de informação. Tenho a felicidade de pessoalmente conhecer um competentíssimo cirurgião, agora com 69 anos, na especialidade de otorrinolaringologia. Da última vez que o visitei, há 4 semanas, actualizou-me com um número por volta das trinta e duas mil e quinhentas cirurgias no seu currículo. Ele não é bom porque já foi bom. Ele não é bom porque lê muito. Ele não é bom porque também publica e dá palestras. Ele é bom porque pratica. E pratica em paralelo com os cargos de direcção que assume. Infelizmente, este paralelo não se vê muito no desenvolvimento de sistemas de informação, e com isto refiro-me ao desequilíbrio na valorização, do mercado das revistas e conferências, e do condicionamento dos investigadores para a escrita de artigos. Produtos, arquitecturas e patentes, são também investigação. Refiro-me

ainda a quem dirige e não programa: se não pratica, não sabe. Se não sabe, não pode superintender.

O terceiro ponto que gostaria de partilhar tem a ver com a segurança, ou melhor, a gravidade de não saber vender a segurança. Vou de novo usar um exemplo de outro domínio, antes de regressar à segurança na informática. Quando tinha três anos os *velhotes* compraram um carro que ainda hoje tenho e estimo. Fui crescendo e fazendo algumas questões. Tinha havido uma crise do petróleo e não havia propriamente carros baratos. Depois havia carros de luxo, e uns excêntricos, lá dos nórdicos, que eram conhecidos por serem seguros. As coisas evoluíram, e muito desse conhecimento democratizou-se, e hoje está presente na generalidade dos veículos. Na área dos sistema de informação, continua-se a olhar a segurança como um luxo. Quem sabe inflaciona o seu preço, quem paga desvaloriza a sua importância. Não havendo um ponto de equilíbrio, são exploradas as fragilidades, por quem, sabendo, e não tendo melhor mercado ou ética, vê oportunidades a crescerem na proporção do aumento de dispositivos e aplicações interligados, numa rede mundial comum. Os construtores de veículos automóveis faziam questão de exibir os *crash-tests*. Por um lado tal aumentava a consciencialização do utilizador e motivava-o a pagar por algo invisível e, que em teoria, tinha uma probabilidade mínima de acontecer. Por outro lado, tal exibia-se como factor diferenciador ou de vanguarda, em relação à concorrência. Um bom gestor de projecto, não tem de destruir nenhum computador, basta recortar (metaforicamente falando), um par de notícias para exemplificar o quão grave pode torna-se, o aligeirar na componente da segurança – em proporção da responsabilidade do que lá trata e do quanto custará recuperar.

Por fim, é do conhecimento geral que o os sistemas electrónicos de votação facilitam uma maior participação dos eleitores; oferecem uma maior acessibilidade, quer a residentes no estrangeiro, quer a cidadãos com necessidades especiais; garantem uma auditabilidade durante e após o processo eleitoral, têm uma melhor eficiência, com uma clara redução de custos; têm uma melhor eficácia, ao evitar erros de contagem manual. Então, porque razão não são utilizados a nível de eleições para órgãos de soberania nacional? Posso adiantar que têm razão, e motivo não é preconceitual. Estou seguro porém que as tecnologias, os protocolos e as sociedades vão encontrar formas de incrementar a participação e a intervenção cívica da sociedade. Da minha parte, e posso falar seguramente em nome da equipa com quem trabalhei, temos a certeza que fizemos o nosso melhor, ultrapassando em muito o que nos foi pedido, com prejuízos a nível financeiro e de valorização profissional.

Bibliografia

[1] Amin Abdollahi, Hossein Roghani-Mamaqani, Mehdi Salami-Kalajahi, and Bahareh Razavi. Encryption and authentication of security patterns by ecofriendly multi-color photoluminescent inks containing oxazolidine-functionalized nanoparticles. *Journal of Colloid and Interface Science*, 580:192.210, 2020.

[2] Luis Abreu and Paulo Morgado. *LINQ com C#*. FCA, 2009.

[3] Ben Adida, Olivier de Marneffe, and Olivier Pereira. Helios voting. = https://vote.heliosvoting.org/.

[4] Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. *EVT/WOTE2009: Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, 2009.

[5] Rachel F. Adler and Raquel Benbunan-Fich. Juggling on a high wire: Multitasking effects on performance. *International Journal of Human-Computer Studies*, 70(2):156 – 168, 2012.

[6] A. Aguiar. Agile software development. Formação FEUP, 2004.

[7] José Almeida, Cécile Baritel-Ruet, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vicent Laporte, Tiago Oliveira, Alley Stoughton, and Strub Pierre-Yves. Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurence implementations of sha-3. *ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, 2019.

[8] Kent Beck. An example of preparatory refactoring. `https://martinfowler.com/articles/preparatory-refactoring-example.html`, 2015. Accessed: 2021-03-02.

[9] Georg Becker. Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-Universität Bochum*, 2008.

[10] Keith Brown and Sesha Mani. Microsoft windows identity foundation (wif) whitepaper for developers. *Microsoft Corporation*, 2008.

[11] Schneier Bruce. *Applied Cryptography - protocols, algorithms and source code in C*. John Wiley & Sonsg, 1996.

[12] Ames Research Center, editor. *Stress, Cognition, and Human Performance: A Literature Review and Conceptual Framework*. NASA, 2004.

[13] Peter Chen. The entity-relationship model -toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[14] World Wide Web Consortium. Latest soap versions. = https://www.w3.org/TR/soap/, 2007.

[15] World Wide Web Consortium. Web services activity. =https://www.w3.org/2002/ws/, 2011.

[16] Microsoft Corporation. Request limits. `https://docs.microsoft.com/en-us/iis/configuration/system.webserver/security/requestfiltering/requestlimits/`, 2016.

[17] Microsoft Corporation. Request limits. `https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.timeout?view=net-5.0`, 2016.

[18] Diário da República, editor. *Regulamento n.º 611/2019, Ordem dos Arquitectos*. Diário da República, Ago 2019.

[19] Christopher J. Date. *An Introduction to Database Systems, 8th edition*. Addison-Wesley, 2004.

[20] docs.microsoft.com. Choosing a datareader or a dataset. `https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-architecture`, 2021. Accessed: 2021-02-06.

[21] Asicioglu Faruk, Tekin Tugba, Ozbek Nil, Ekim Filiz, and Ozcan Mustafa. Prepared disappearing ink and deciphering of documents. *Journal of Forensic Sciences*, 64:1898–1905, 2019.

[22] Roy Fielding. Representational state transfer. `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`, 2000.

[23] Internet Engineering Task Force. The javascript object notation (json) data interchange format. = https://datatracker.ietf.org/doc/html/rfc7159, 2014.

[24] Martin Fowler. Continuous integration. `https://www.martinfowler.com/articles/continuousIntegration.html`, 2006. Accessed: 2020-12-20.

[25] ghostinspector.com. Automated selenium testing made easy. `https://ghostinspector.com/docs/`, 2020. Accessed: 2020-12-23.

[26] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4), 1983.

[27] Regina Hebit et al. A web service architecture for decentralised identity and attribute-based access control. In IEEE, editor, *International Conference on Web Services*, 2009.

[28] Pedro Rangel Henriques and José Carlos Ramalho. *XML & XSD - da teoria à prática*. FCA, 2002.

[29] Ge Huangyi et al. *Koinonia: Verifiable E-Voting with Long-term Privacy*. Annual Computer Security Applications Conference, 2019.

[30] Andy Hunt and Dave Thomas. *Pragmatic Unit Testing – in C# with NUnit*. The Pragmatic Programmers, LLC, 2004.

[31] DigiCert Inc. Beginner's guide to tls/ssl certificates. = https://https://www.digicert.com/resources/beginners-guide-to-tls-ssl-certificates-whitepaper-en-2019.pdf, 2019.

[32] ISO/IEC. Information technology - open systems interconnection. = http://standards.iso.org/ittf/PubliclyAvailableStandards, 1996.

[33] JSON.org. Introducing json. = https://www.json.org/.

[34] Kaindl H. Kajko-Mattsson M., Aguiar A. et al. Long-term perspective of agile methods. *2009 Fourth International Conference on SW Engineering Advances*, 10(1109), 2009.

[35] L. L. Kemmeren, D. J. F. Van Schaik, J. H. Smit, J. Ruwaard, A. Rocha, M. R. Henriques, D. D. Ebert, I. Titzler, JB. Hazo, M. Dorsey, K. Zukowska, and H. Riper. Unraveling the black box: Exploring usage patterns of a blended treatment for depression in a multicenter study. *JMIR Mental Health*, 6(7), 2019.

[36] Jamie Kurtz. *ASP.NET MVC4 and the Web API.* APRESS, Springer Science+Business Media New York, 2013.

[37] Mitch Lacey. *The SCRUM field guide, 2nd edition.* Addison-Wesley, 2016.

[38] Peter Larsen. *VDM-Tools - the IFAD VDM-SL Language.*

[39] Fowler M. et al. Chrysler goes to extremes. Technical report, www.DistributedComputing.com, 1998.

[40] Microsoft. Entity framework overview. `https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview`, 2021. Accessed: 2021-02-08.

[41] Microsoft. Introduction to identity on asp.net core. `https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0`, 2021. Accessed: 2021-03-02.

[42] Masashi Mizoguchi, Takahiro Iida, and Toru Irie. Optimization of automated executions based on integration test configurations of embedded software. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 358–363, 2020.

[43] Richar Montgomery, Alexey Pokrovskiy, and Benny Sudakov. *Decompositions into spanning rainbow structures.* University of Birmingham, 2019.

[44] JS Foundation Project. Introducing appium. `http://appium.io/`, 2020. Accessed: 2020-12-23.

[45] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems, 3rd edition.* McGraw-Hill, 2002.

[46] Trygve Reenskaug and James O. Coplien. The dci architecture: A new vision of object-oriented programming. = https://www.artima.com/articles/the-dci-architecture-a-new-vision-of-object-oriented-programming, 2009.

[47] Leonard Richardson. Richardson maturity model. `https://martinfowler.com/articles/richardsonMaturityModel.html`, 2010.

[48] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1978.

[49] Artur Rocha, Ricardo Henriques, et al. Ict4depression: Service oriented architecture applied to the treatment of depression. In Computer-Based Medical Systems (CBMS), editor, *25th International Symposium*, 2012.

[50] Walker Royce. *Software Project Management*. Addison-Wesley, 1998.

[51] Rui Sá. *Introdução às Redes de Telecomunicações*. FCA, 2016.

[52] Ahmed Seffah, Taleb Mohamed, Halima Habieb-Mammar, and Alain Abran. Reconciling usability and interactive system architecture using patterns. *The Journal of Systems and Software*, 81:1845–1852, 2008.

[53] Quan Sheng et al. Compatibility checking of heterogeneous web service policies using vdm++. In IEEE, editor, *Congress on Services - 1*, 2009.

[54] The Internet Society. Http status code definitions. `https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`, 1999.

[55] StackExchange. = https://security.stackexchange.com/questions/183179/what-is-rsa-oaep-rsa-pss-in-simple-terms/183330♯183330, 2018.

[56] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markkov. The first collision for full sha-1. *Centrum Wiskunde & Informatica, Amsterdam*, 2017.

[57] Cvetkovic. Stevica and Dragan Jankovin. A comparative study and performance of orm tools in a .net environment. In *Lecture Notes in Computer Science*, volume 6348, pages 147–158, 2010.

[58] Marian Stoica, Marinela Mircea, and Bogdan Ghilic-Micu. Software development: Agile vs. traditional. *Informatica Economică*, 17(4), 2013.

[59] Sven Strickroth et al. Tortoisegit. `https://tortoisesvn.net`. Accessed: 2020-12-15.

[60] Andrew Tanenbaum. *Computer Networks, Fourth Edition*. Prentice Hall, 2003.

[61] Andrew Troelsen. *Pro C# 2008 and the .NET 3.5 Platform*. APRESS, 2008.

[62] Pepijn van de Ven, Hugh O'Brien, Ricardo Henriques, Michel Klein, Rachel Msetfi, John Nelson, Artur Rocha, Jeroen Ruwaard, Donal O'Sullivan, and Heleen Riper. Ultemat: A mobile framework for smart ecological momentary assessment and interventions. *Internet Interventions*, 9:74–81, 2017.

[63] State Library Victoria. Researching your ancestors from great britain and ireland. = https://guides.slv.vic.gov.au/britishislesancestors/electoral, 2021.

[64] Carsten Willems. Internals of windows memory management (not only) for malware analysis. Technical report, University of Mannheim, Germany, 2011.

[65] Runhua Xu and James Joshi. Trustworthy and transparent third-party authority. *ACM Trans. Internet Technol.*, 2020.

[66] Robert Yerkes and John Dodson. *The Relation of Strength of Stimulus to Rapidity of Habit-Formation*. Jornal of Comparative Neurology and Psychology, 1908.

[67] Dmitriy Zaporozhets. Gitlab. `https://about.gitlab.com`. Accessed: 2020-12-15.