Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Projecto e Seminário

Teamworks

a solution for project planning, managing and collaboration

Alunos

João Santos 30071 (a30071@alunos.isel.pt)

Filipe Pinheiro 30239 (a30239@alunos.isel.pt)

Orientador

João Pedro Patriarca (jpatri@cc.isel.ipl.pt)

13 de Agosto de 2012

Índice

1 Introdução

Actualmente é frequente qualquer pessoa estar envolvida em projectos, num contexto profissional e/ou pessoal, em que o aproveitamento do tempo despendido em cada um é uma preocupação constante. Esta preocupação pode ser atenuada com uma gestão eficaz desses projectos. A gestão de um projecto não se resume à organização de tempo, é também necessário, para uma gestão eficaz, planeamento, organização, controlo de recursos e monitorização.

Este projecto tem como objectivo resolver esse problema através de uma plataforma de apoio à gestão de projectos. A motivação para a sua realização surgiu porque as soluções de *software* existentes ou não oferecem todas as funcionalidades necessárias ou são demasiado complexas.

Para além da gestão do projecto é também importante gerir as pessoas que dele fazem parte. A comunicação entre membros de um projecto pode ter um grande impacto na qualidade e velocidade com que o produto final é entregue. Desta forma a plataforma desenvolvida, para além da gestão de projectos, também é capaz de gerir os seus membros, facilitando a comunicação e partilha de informação entre eles.

1.1 Soluções Existentes

As soluções existentes que pretendem agilizar a gestão de projectos e podem ser encontradas sobre a forma de aplicações desktop ou web.

As aplicações desktop são usadas em ambiente local limitando a interacção e partilha de resultados com outras pessoas do projecto. Um exemplo deste tipo de aplicações é o *Microsoft Project* [1] usado para o planeamento de tarefas, estimar a sua duração e atribuir-lhe recursos.

Ao contrário das aplicações desktop todas as interacções com uma aplicação web ficam alojadas na *cloud* tornando a informação acessível a todos os membros do projecto. Há uma grande variedade de aplicações web disponíveis, entre as quais, se destacam as aplicações *basecamp* [2], *freckle* [3] e *LiquidPlanner* [4].

A aplicação web *basecamp* foca as suas funcionalidades na partilha e armazenamento de informação dos projectos criados. Tem ainda gestão de actividades, *todos* e eventos.

A aplicação web *freckle* tem como funcionalidades principais o registo de horas e monitorização de tempo despendido por tarefa.

A aplicação web *LiquidPlanner* oferece as funcionalidades das anteriores e ainda o planeamento de tarefas mas com uma interface mais complexa que as soluções anteriores, o que aumenta o tempo despendido na gestão do projecto.

1.2 Solução Teamworks

A solução *Teamworks* é composta por uma aplicação web e por uma web Api. A Api expõe todas as funcionalidades da plataforma permitindo o seu acesso

através de diversos clientes (e.g. browser, aplicações móveis, aplicações desktop). A aplicação web assenta sobre a Api e tem uma interface simples com o objectivo de minimizar o tempo utilizado na plataforma.

As funcionalidades disponibilizadas permitem a gestão de projectos e pessoas. Para a gestão de projectos é possível planear tarefas, estimar a sua duração, atribuir-lhes responsáveis e monitorizar o estado global do projecto. Para gerir pessoas estas podem ver as tarefas que têm a realizar, registar o tempo despendido e para promover a colaboração partilhar informações e debater soluções.

2 Descrição Geral

Como indicado anteriormente, a solução *Teamworks*, tem como objectivo disponibilizar funcionalidades para a gestão de projectos e pessoas. Para isso a cada projecto estão associadas as seguintes funcionalidades:

- Possibilidade de criar actividades, gerir o tempo estimado da sua execução, atribuir responsáveis pela sua realização e disponibilizar uma lista a cada utilizador para que possa decidir a que actividades dar prioridade e organizar o seu tempo.
- Possibilidade de cada utilizador registar o tempo que despendeu em determinada actividade. Esta funcionalidade é importante para controlar se as estimativas estão correctas e se o desenvolvimento do projecto está a correr como esperado.
- Ferramentas para monitorização do estado do projecto.

Para promover a colaboração existe ainda a possibilidade dos utilizadores terem uma área de "debate" onde partilham informação, trocam ideias e debatem soluções relacionadas com o projecto ou com tarefas.

Para além das funcionalidades enunciadas o acesso à informação dos utilizadores e projectos deve estar disponível aos utilizadores com quem tenha sido partilhada.

2.1 Casos de Utilização

A autenticação na plataforma é obrigatória para a sua utilização, sendo que as únicas acções possíveis a um utilizador anónimo são o registo e a autenticação, como indicado na figura 1.

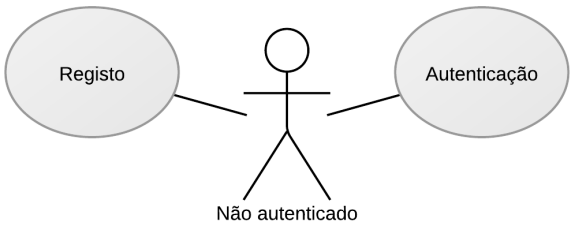

Figura 1: Caso de utilização de utilizador não autenticado.

Depois de autenticado o utilizador tem a opção de criar um projecto ou alterar um projecto já existente. A um projecto já existente é possível adicionar e remover pessoas, adicionar actividades ou apagar o projecto. Ao criar um projecto o utilizador fica associado a este e tem a possibilidade de fazer todas as acções enunciadas

Ao criar uma actividade são disponibilizadas novas acções ao utilizador. O utilizador pode, sobre a actividade criada, criar novas entradas de registo do tempo despendido na sua realização, adicionar e remover pessoas e apagá-la.

De forma a promover a troca e partilha de informação é possível aos utilizadores criar e aceder aos debates relacionados com os projectos e/ou tarefas em que estão envolvidos.

Os casos de utilização enunciados podem ser observados na figura 2

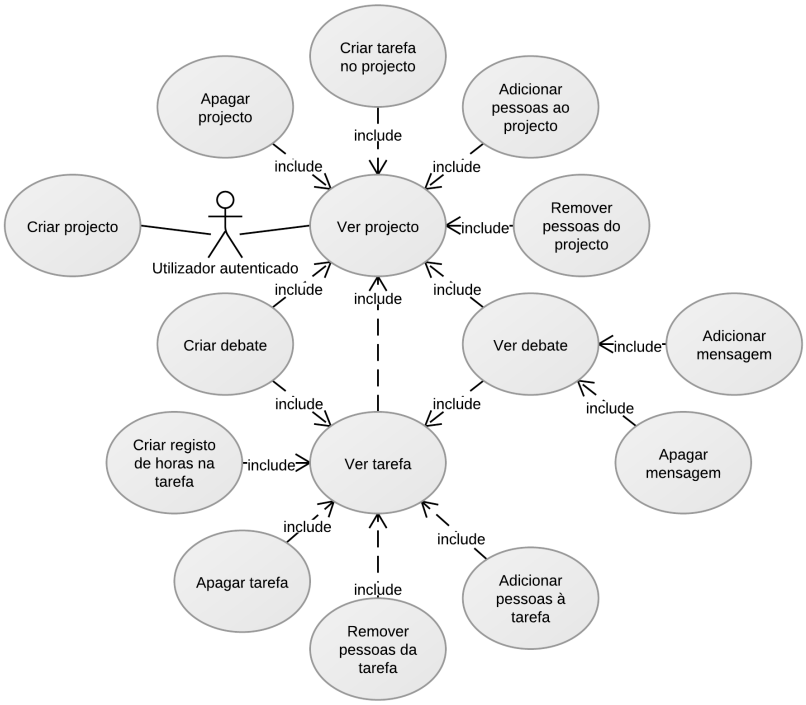

Figura 2: Casos de utilização de um utilizador autenticado

O acesso á plataforma pode ser feito através de um browser ou de um cliente da Api.

2.2 Arquitectura

Na arquitectura da solução destacam-se quatro componentes, a aplicação web, a Api, a camada de serviços e a de dados, todos eles implementados utilizando a *framework* .NET [7]. A aplicação web utiliza processamento do lado do cliente para complementar a implementação do servidor.

A aplicação web e a Api são implementadas utilizando a *framework* ASP.NET [8] e expõem as funcionalidades da plataforma através do protocolo HTTP. A camada de serviços é responsável por toda a lógica aplicacional e a camada de dados tem como responsabilidade persistir os dados e disponibilizá-los quando pedidos. Para a persistência dos dados é usada a camada de dados.

A interacção dos componentes é a seguinte: a aplicação web e a Api usam a camada de serviços para responder aos pedidos que lhes são feitos; a camada de serviços envia e obtém dados da camada de dados para implementar a sua lógica; e a camada de dados é responsável pela comunicação com a base de dados. A figura 3 demonstra esta interacção.

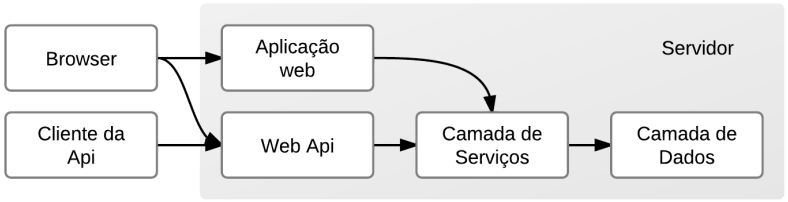

Figura 3: Arquitectura da plataforma *Teamworks*

Alguns dos problemas não directamente relacionadas com a plataforma são resolvidos usando projectos *opensource* que serão indicados quando se justificar.

3 Domínio

Para descrever o modelo de domínio foi utilizada uma abordagem *Domain-Driven Design* [5] (descrita no anexo A). Como base para a elaboração do modelo de domínio foram usadas as características e funcionalidades apresentadas na descrição geral (ver secção 2).

3.1 Modelo de Domínio

As entidades raiz identificadas são **pessoa** (`Person`) e **projecto** (`Project`). Cada pessoa pode estar envolvida em vários projectos, e um projecto pode ter várias pessoas envolvidas.

A entidade **pessoa** é representada pelo nome de utilizador, *email* e *password*. O *email* é usado para comunicar com a pessoa e os outros dois atributos para autenticar o utilizador. A entidade **projecto** agrega as pessoas que lhe estão associadas sendo possível definir **actividades** que, como **pessoa** e **projecto**, são entidades do domínio.

Uma **actividade** (`Activity`) tem nome e descrição e pode também ter várias pessoas associadas. Para além destes atributos, tem ainda o tempo estimado para a sua realização (e.g. número de horas), a data prevista de conclusão e **registos de tempo** (`Timelog`) despendido na sua realização, que podem ser adicionados pelas **pessoas** associadas.

Sobre os projectos e tarefas é ainda possível criar **debates** (`Discussion`), visíveis apenas a **pessoas** que lhes estão associadas, onde é possível criar **mensagens** (`Message`).

As entidades e as suas relações estão representadas no diagrama de classes da figura 4.

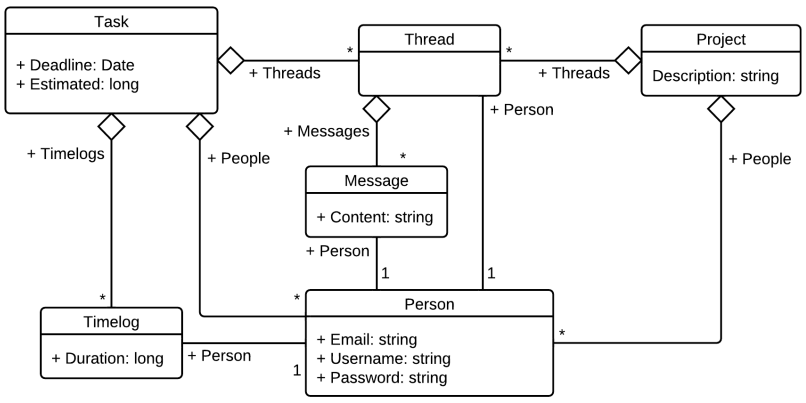

Figura 4: Diagrama de relação entre os objectos de domínio.

Depois de analisadas as relações e responsabilidades de cada entidade foram caracterizados como agregados os objectos de domínio **projecto** e **pessoa**. Os

objectos **actividade** e **debate** são considerados entidades porque são identificados univocamente. A **actividade** é entidade do agregado projecto e o **debate** pertence tanto ao agregado projecto como à entidade actividade. O **registo de tempo** e as **mensagens** são *value object* e inseridos nas entidades que os referenciam.

3.2 Segurança

Por questões de segurança todas as acções na infra-estrutura têm de ser feitas por utilizadores autenticados sendo por isso necessário disponibilizar forma dos utilizadores se registarem. Para manter a privacidade dos dados tem ainda de haver, aliado à autenticação, políticas de acesso e autorização.

No registo o utilizador indica o email, utilizado como meio de comunicação; o nome de utilizador para o identificar; e a password, que em conjunto com o nome de utilizador, é usada para autenticar o utilizador. Esta informação é persistida na base de dados, à excepção da *password* da qual é apenas persistido o *hash*. O *hash* da password é gerado usando um algoritmo de dispersão (*SHA-256*) que tem como entrada a concatenação da password com um *salt*[1] aleatório.

A autenticação é feita usando o nome de utilizador e a *password* e é válida se o resultado da função de dispersão, usada no registo para a *password* for igual ao obtido usando os dados inseridos pelo utilizador. A função de dispersão, na autenticação, tem como parâmetro de entrada a concatenação da *password* inserida com o *salt* presente na instância de *Person* obtida.

A política de acesso é definida individualmente por cada entidade. Sendo que um utilizador só lhe pode aceder se tiver sido criada por si ou se lhe tiverem atribuído permissões.

[1] O *salt* é utilizado para criar aleatoriedade na *password*. Desta maneira o *hash* gerado para a mesma *password* é diferente por utilizador.

4 Dados

Para a realização deste projecto foi usada a base de dados de documentos RavenDB, uma base de dados transaccional, *open-source* e implementada sobre a *framework* .NET. Esta base de dados é composta por um servidor e um cliente e os dados são guardados sem *schema* rígido em documentos no formato JSON.

Na figura 5 pode observar-se a interacção da plataforma com o cliente RavenDB.

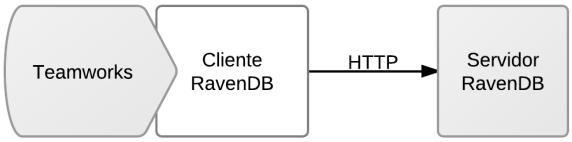

Figura 5: Interacção da plataforma Teamworks com a base de dados.

A responsabilidade de persistir os dados é do servidor que diponibiliza uma Api ReSTful para todas as comunicações de acesso aos dados.

O cliente expõe as funcionalidades do servidor e permite a interacção através de código .NET . Os dados enviados e recebidos do cliente são instâncias de classes *POCO*s (*Plain Old CLR Object*) o que simplifica a sua utilização.

```
1  var person = new Person {
2                  Email = "johndoe@world.com"
3                  Username = "johndoe",
4                  Password = "wh0 am i?"
5                }
6
7  using(var session = store.OpenSession()) {
8    session.Store(person);
9    session.SaveChanges();
10 }
11
12 using(var session = store.OpenSession()) {
13   var entity = session.Query<Person>()
14     .Where(p => p.Username == "johndoe")
15     .Single();
16
17   entity.Username = "doejohn";
18   session.SaveChanges();
19 }
```

Lista 1: Utilização do cliente RavenDB.

A lista 1 demonstra a utilização do cliente. Pode observar-se a utilização de *POCO*s e do padrão *Unit of Work* [6, pp. 184-194] pois todas as alterações

feitas ao cliente são persistidas na base de dados numa única transacção quando é chamado o método `SaveChanges`. A variável `store` define a configuração do cliente, a comunicação com o servidor e todos os mecanismos da base de dados.

4.1 Modelo de Dados

Devido à escolha de uma base de dados de documentos a forma de modelar os dados da aplicação tem de ser ajustada.
Uma característica a ter em conta neste tipo de base de dados é a impossibilidade de fazer operações de `JOIN`, o que implica que um documento deve guardar toda a informação necessária da entidade que representa.

A abordagem aconselhada para a modelação de dados neste tipo de base de dados é o uso do padrão *Aggregate* [5, pp. 126-127] para a escolha de que informação fica em cada documento. O padrão define um agregado como um grupo de objectos tratados como um só, tendo em conta alterações no seu conteúdo. As referências externas estão limitadas à raiz do agregado, que controla todas as alterações aos objectos contidos nos seus limites. Como na definição do modelo de domínio foi usada uma abordagem *Domain Driven Design* os agregados identificados são guardados como documentos independentes. A única excepção a esta regra é a entidade actividade que tem um documento próprio. Esta decisão deve-se à possibilidade de um projecto ter várias actividades associadas e para evitar que o documento atinja dimensões elevadas.

A figura 6 representa o modelo de dados da solução, através de um diagrama de classes UML.

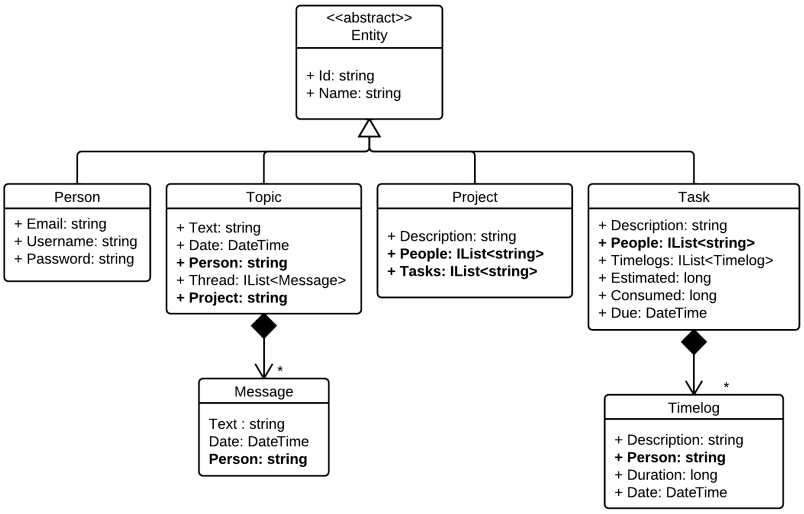

Figura 6: Diagrama UML de classes do modelo de dados.

A autorização na plataforma utiliza um *bundle*[2] do RavenDB. O *bundle* permite fazer a gestão de obtenção, alteração e remoção de documentos baseado

[2]Módulo de extensão das funcionalidades da base de dados - Ver anexo B.3.3.

no utilizador. Este *bundle* define quatro intervenientes no processo de autorização: o utilizador (`AuthorizationUser`), o *role* (`AuthorizationRole`), a operação que o utilizador pode fazer (`OperationPermission`) e a permissão necessária para aceder ao documento (`DocumentPermission`).

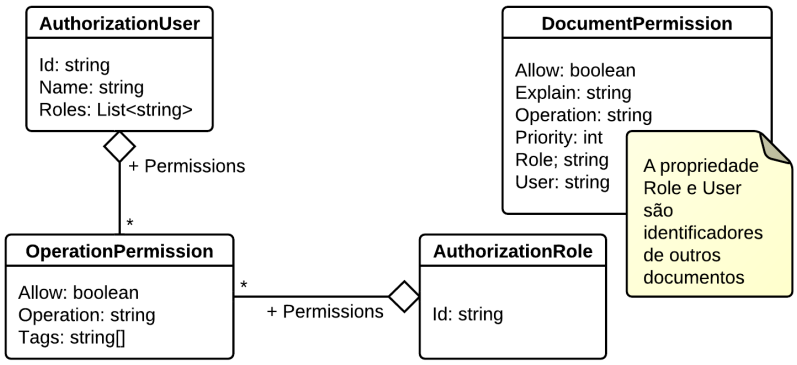

Figura 7: Diagrama UML de autorização.

Para um utilizador aceder a um documento tem de lhe estar associado, pertencer a um *role* associado ao documento, ter a operação exigida pelo documento ou pertencer a um *role* que a tenha. No entanto este módulo permite que seja usado um tipo personalizado para representação do utilizador desde que defina as mesmas propriedades que a implementação disponibilizada pelo Raven (`AuthorizationUser`). No modelo de dados da plataforma o utilizador é substituído por `Person` e o *role* não é utilizado.

5 Serviços

Os serviços disponibilizados pela infra-estrutura encontram-se no *namespace* `Teamworks.Core` e incluem serviços de acesso a dados e autenticação.

5.1 Acesso a Dados

É comum para acesso a dados a utilização do padrão *Repository* [6, pp. 322-327]. Este padrão encapsula a lógica de persistência dos objectos de domínio de forma a desacoplar o modelo de domínio da lógica de acesso a dados. Na solução *Teamworks* a implementação do padrão *Repository* é feita pelo cliente *RavenDB*

A inicialização e configuração do cliente (`IDocumentStore`) está presente na propriedade `Database` da classe `Global` e a sua inicialização deve ser feita no ínicio da aplicação.

5.2 Segurança

A plataforma impõe as políticas de acesso em conjunto com o *Authorization bundle*. A autorização é feita quando se tenta aceder a um documento. Se um utilizador tentar aceder a um documento e não tiver permissões para o fazer é lançada uma excepção.

A configuração do cliente, para que este valide se é possível interagir com o documento, é feita utilizando métodos de extensão presentes no ficheiro `Raven.Client.Authorization.dll`.

6 Web Api

A Api assenta sobre modelo de arquitectura ReSTful e todos os objectos de domínio da aplicação são considerados recursos com URL próprio. A comunicação é feita utilizando o protocolo HTTP.

O processamento do pedido HTTP é iniciado no *pipeline* da *framework* ASP.NET Web Api. Todos os elementos do *pipeline* derivam da classe `HttpMessageHandler`, o primeiro elemento é do tipo `HttpServer` e o último elemento do tipo `HttpControllerDispatcher` que tem como responsabilidade, utilizando a tabela de *routing*, obter e chamar o *controller* respons´´avel por processar o pedido.

É possível extender o *pipeline* adicionando novos elementos antes do processamento do *controller* tendo estes que derivar de `DelegatingHandler`(que deriva de `HttpMessageHandler`). A classe `DelegatingHandler` usa o padrão *delegator* que define que cada instancia tem de referenciar uma instancia de `DelegatingHandler`. Desta forma cria-se uma cadeia de *handlers* que sequencialmente vão passando a mensagem ao *handler* seguinte.

Na figura 8 estão representados os elementos adicionados ao *pipeline* e a ordem por que são executados, `RavenSessionHandler` e `BasicAuthenticationHandler` implementados por classes com o mesmo nome.

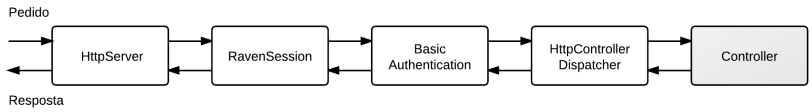

Figura 8: Processamento do pedido na Web Api.

- `RavenSession`

  O elemento `RavenSession` instancia uma sessão de acesso à base de dados e adiciona-a às propriedades do pedido, lista 2.

  Na resposta este elemento verifica se não ocorreu nenhum erro no processamento do pedido, persistindo as alterações em caso de sucesso, como demonstra a lista 3.

```
1 var session = Global.Database.OpenSession();
2 request.Properties[Application.Keys.RavenDbSessionKey] =
     session;
```

Lista 2: Processamento do pedido da classe ‘RavenSessionHandler‘.

```
1 using (session) {
2   if (session != null && t.Result.IsSuccessStatusCode) {
3     session.SaveChanges();
```

```
4     }
5 }
```

Lista 3: Processamento da resposta da classe 'RavenSessionHandler'.

- `BasicAuthentication`

  No processamento do pedido o elemento `BasicAuthentication` verifica se o *header* de nome `Authorization` tem como esquema de autenticação *Basic* e converte as credenciais para a forma original, `nome-de-utilizador:password`. A validação das credenciais enviadas para o servidor utiliza o processo descrito no domínio (ver secção 3).

  Na resposta se o código for `401 Unhautorized` este elemento adiciona o *header* de autenticação para indicar que aceita autenticação do tipo *Basic*.

O último elemento do *pipeline* é da classe `HttpControllerDispatcher`. A classe `HttpControllerDispatcher` tem como responsabilidade, utilizando a tabela de *routing*, instanciar o *controller* para processar a mensagem e gerar a resposta.

6.1 *Controllers*

No fim do *pipeline* é criada a instancia de um *controller* e antes de invocar a *action* com a responsabilidade de processar o pedido

O `HttpControllerDispatcher` instancia o *controller* e realiza processamento antes de invocar a *action* correspondente ao método HTTP da mensagem. Depois de selecionada a *action* são executados of filtros, é feito o model binding e só depois a *action* é invocada. Depois de executada a *action* é criado o conteúdo da resposta com base nos *formatters* disponíveis e na negociação de conteúdo com o cliente.

No contexto de um controller é possível definir processamento para além daquele realizado pela *action* selecionada. O processamento extra pode ser feito através de filtros. Filtros que podem ser globais, associados ao *controller* ou a uma *action* e derivam de `ActionFilterAttribute`. Os filtros globais são registados no inicio da aplicação e os restantes como atributos aplicados ao *controller* ou *action* dependendo do contexto.

Os atributos definidos na implementação da infra-estrutura são os seguintes:

- `MappingExceptionFilterAttribute`

  O `MappingExceptionFilterAttribute` deriva de `ExceptionFilterAttribute`, um tipo especifico de `ActionFilterAttribute`, e para além de redefinir o método `OnException` tem uma propriedade `Mappings` que relaciona um tipo com uma regra(`Rule`). A regra define o código associado e se a mensagem da excepção é incluída no corpo da resposta.

  Como pode ser observado na figura 9 o método redefinido obtem a excepção e dependendo do seu tipo cria a resposta adequada. A resposta criada

se a excepção for uma `HttpException` tem o código associado á excepção. Se for de um tipo presente na propriedade `Mappings` a resposta tem o código associado e o corpo definido na regra registada. Por fim se não for do tipo `HttpResponseException` a resposta tem o código `500 Internal Server Error`. O tipo `HttpResponseException` não é considerado porque é processado pela classe base.

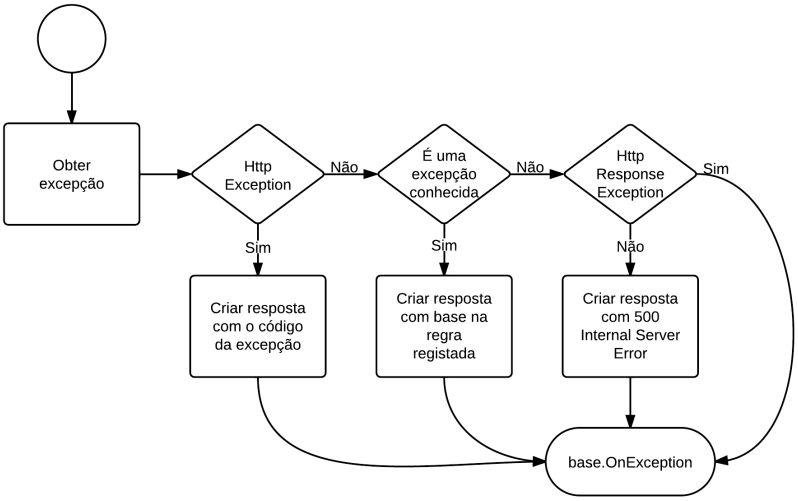

Figura 9: Fluxo de processamento de MappingExceptionFilterAttribute.

- `ModelStateFilterAttribute`

O processamento do controller

- `SecureForFilterAttribute`
- `RavenSessionFilterAttribute`

  Para abstrair as *actions* da obtenção da sessão foi criada a classe `RavenApiController` que disponibiliza a propriedade `DbSession`, que retorna uma sessão para acesso à base de dados. O propriedade `DbSession` é afectada com a instancia passada por parâmetro no construtor do *controller*.

  O atributo `RavenSessionFilterAttribute`

7 Aplicação Web

A aplicação web disponibiliza uma interface para interacção com a infra-estrutura, implementada usando a *framework* ASP.NET MVC [11].

A aplicação web disponibiliza *controllers*, cujas *actions* têm como resposta aos pedidos uma página web com código *javascript* que tornam o browser um cliente da Api. As páginas web retornadas utilizam as *frameworks javascript* jQuery [12] e knockout [13] para interacção com o utilizador.

A componente visual é conseguida usando HTML5, CSS3 e o *kit* Twitter Bootstrap [14].

A autenticação na aplicação web é feita através de um formulário onde o utilizador insere o nome de utilizador e a *password*. A validação dos dados inseridos é feita de forma semelhante à utilizada na Api com a diferença que na aplicação web é usada uma cookie para manter o utilizador autenticado nos pedidos subsequentes. Para isso são usadas funcionalidades do modo de autenticação *forms* da *framework* ASP.NET.

As classes que expõem as funcionalidades necessárias são a classe `FormsAuthentication`, usada para gerir o cookie de autenticação[3], e o módulo `FormsAuthenticationModule` para manter o utilizador autenticado.

Depois de validar os dados do utilizador com sucesso a cookie é colocada na resposta usando a classe `FormsAuthentication`. A cookie tem como valor o identificador da instância de *Person* que foi obtida da base de dados no processo de autenticação.

O utilizador mantém a identidade dos pedidos anteriores pois o módulo `FormsAuthenticationModule`, em cada pedido, coloca o valor da *cookie* na propriedade `User` do pedido. A acção do módulo é complementada pelo filtro `FormsAuthenticationAttribute` que substitui o `IIdentity` do pedido por um `PersonIdentity` que disponibiliza a entidade `Person` do utilizador autenticado ao código executado nas *actions*.

Para que um utilizador quando autenticado na aplicação web também esteja autenticado na Api foi acrescentado ao *pipeline* da Api a classe `FormsAuthenticationHandler`. Esta implementação de `DelegatingHandler` verifica se o `IPricipal` do pedido está autenticado e se contiver uma instância de `PersonIdentity` coloca a `Person` correspondente na colecção `Properties` utilizando a chave `HttpPropertyKeys.UserPrincipalKey`.

Na aplicação web um utilizador está autorizado a aceder a qualquer um dos *controllers* que interagem com a Api desde que autenticado.

7.1 Cliente da Api

A forma como foi desenvolvida a aplicação web torna-a um cliente da Api da plataforma. Este comportamento é conseguido através da utilização da framework javascript knockout e AJAX. Esta framework é usada para fazer actualização

[3] A cookie usada tem o nome *.tw_auth*

de elementos da página web com a informação do servidor, respondendo a interacções com o utilizador. Neste componente é usado o padrão MVVM (**M**odel-**V**iew-**V**iew**M**odel) para tornar as páginas web mais dinâmicas e melhorando a experiência do utilizador.

Este comportamento é conseguido definindo, em javascript, um *View Model* que representa do dados retornados pela Api. Estes View Models definem propriedades observáveis por elementos HTML, através de javascript. Através desta ligação é possivel o valor do elemento HTML seja alterado consoante o valor do atributo que observa e vice versa.

8 Conclusão

A solução desenvolvida permite aos utilizadores fazer diversas acções sobre a plataforma, como a criação e gestão de projectos, tarefas e debates. No entanto ainda existe trabalho que é necessário fazer para que a plataforma atinja os níveis a que nos propusemos.

No desenvolvimento deste projecto foram usadas técnicas e tecnologias com as quais não estávamos familiarizados, havendo por isso um período de aprendizagem. Como exemplo temos as bases de dados de documentos, *domain driven design*, *RavenDB*, *knockout* e *web Api*.

Este desconhecimento inicial, principalmente das bases de dados de documentos, *domain driven design* e *RavenDB*, condicionou o desenvolvimento do modelo de dados, tornando-o um processo iterativo. Durante este processo, e com a aprendizagem feita sobre estes temas, o modelo de dados foi alterado até chegar ao estado actual.

8.1 Trabalho Futuro

A versão actual da plataforma permite a edição dos seus dados de uma forma muito básica. Até à data de entrega da versão final pretende-se implementar funcionalidades que complementem as que já estão disponíveis. As funcionalidades a implementar são:

- Autorização

  No decorrer do projecto foi detectado um erro na implementação do *bundle* de autorização disponibilizado pelo RavenDB[4] o erro já foi corrigido.

- Planeamento de tarefas

  Adicionar à gestão de tarefas a possibilidade de lhes atribuir prioridades e que dependam umas das outras. Desta forma é melhorado o controlo e organização das tarefas de um projecto.

- Análise e Monitorização

  Pretende-se implementar indicadores para análise de informação sobre o projecto, as suas tarefas e utilizadores para que o responsável por um projecto seja capaz de avaliar qual o estado em que este se encontra.

- *Dashboard*

  Deverá estar disponível aos utilizadores informação sobre quais os projectos e respectivas tarefas que lhe estão atribuídas para um período de tempo (e.g. semana actual).

Além de novas funcionalidades existem aspectos da solução que têm de ser melhorados. A cobertura dos testes unitários deve ser alargada tanto na Api, como na aplicação web e em todos os serviços.

[4] O *topic* http://goo.gl/FpIoq no Google Groups do RavenDB detalha o erro e a solução.

A forma de autenticação usada na api web é pouco segura por isso devem ser estudadas alternativas, como a utilização do esquema HMAC (Hash Message Authentication Code) ou Oauth.

Referências

[1] Microsoft, Microsoft Project [Online - acedido a 6 de Junho de 2012] http://www.microsoft.com/project/

[2] 37Signals, Basecamp [Online - acedido a 6 de Junho de 2012] http://www.basecamp.com

[3] Slash7, Freckle [Online - acedido a 6 de Junho de 2012] http://letsfreckle.com/

[4] LiquidPlanner, LiquidPlanner [Online - acedido a 6 de Junho de 2012] http://www.liquidplanner.com/

[5] Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, 2011

[6] Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2009

[7] Microsoft, .NET framework [Online - acedido a 6 de Junho de 2012] http://www.microsoft.com/net

[8] Microsoft, ASP.NET [Online - acedido a 6 de Junho de 2012] http://www.asp.net

[9] Microsoft, Linq [Online - acedido a 6 de Junho de 2012] http://msdn.microsoft.com/en-us/library/bb308959.aspx

[10] Hibernating Rhinos, RavenDB [Online - acedido a 15 de Junho de 2012] http://ravendb.net/

[11] Microsoft, ASP.NET MVC [Online - acedido a 15 de Junho de 2012] http://www.asp.net/mvc

[12] The jQuery Foundation, jQuery [Online - acedido a 15 de Junho de 2012] http://jquery.com

[13] Knockoutsjs, Knockout [Online - acedido a 15 de Junho de 2012] http://knockoutjs.com/

[14] Twitter, Twitter Bootstrap [Online - acedido a 15 de Junho de 2012] http://twitter.github.com/bootstrap/

[15] Todd Hoff, Drop ACID and think about data [Online - acedido a 15 de Junho de 2012], http://highscalability.com/drop-acid-and-think-about-data

[16] Amazon, Dynamo: Amazon's Highly Available Key-value Store [Online - acedido a 15 de Junho de 2012], http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

[17] Ayende Rahien, That No SQL Thing - Key/Value stores [Online - acedido a 15 de Junho de 2012], http://ayende.com/blog/4449/that-no-sql-thing-key-value-s

[18] Ayende Rahien, That No SQL Thing - Document Databases [Online - acedido a 15 de Junho de 2012], http://ayende.com/blog/4459/that-no-sql-thing-document-databases

[19] IBM, Differences between a document-oriented and a relational database [Online - acedido a 15 de Junho de 2012], http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html#N10062

A Domain Driven Design

Esta abordagem defende que o foco principal no desenvolvimento de software é o modelo de domínio do problema.

O modelo de domínio é o conjunto de vários termos, diagramas e conceitos que representam a informação e o comportamento da aplicação face ao problema. Para representar o modelo de domínio podem ser usados diagramas UML, texto detalhado, esquemas entidade-associação, use cases, etc. Um aspecto importante é compreensão do modelo de domínio por todos os intervenientes no projecto (e.g. arquitectos de software, programadores, cliente). Aos elementos do modelo de domínio dá-se o nome de objectos de domínio.

Alguns objectos de domínio transversais a qualquer projecto são as entidades, os *value objects*, os agregados, as fábricas e os repositórios.

Uma entidade representa um objecto do modelo possível de identificar únivocamente em todo o seu tempo de vida na aplicação.

Um *value object*, assim como uma entidade, é representado pelas suas características e atributos mas não tem identidade no sistema, ou seja *value objects* com os mesmas características e atributos são considerados iguais.

No modelo de domínio quando um grupo de objectos é tratado como uma unidade, no que diz respeito á informação que estes representam, são considerados agregados. Um agregado define um limite e tem como raiz uma entidade. É através dessa entidade que os outros elementos do agregado são acedidos.

As fábricas e os repositórios são usados para gerir o tempo de vida das entidades. As fábricas são usadas na criação e por vezes para abstrair o tipo concreto do objecto criado. Os repositórios são mecanismos que encapsulam a obtenção, alteração e persistência dos dados no sistema.

B NoSQL

As bases de dados NoSQL diferem do modelo relacional porque são tipicamente desenhadas para escalar horizontalmente e podem ter as seguintes características:

- Não têm *schema* fixo;
- Não suportam operações de `JOIN`;
- Suportam o conceito de BASE[5];
- Suportam o conceito de CAP[^CAP];

Embora algumas destas características possam parecer negativas, existem vantagens na utilização deste tipo de base de dados dependendo da sua categoria [15]. Em NoSQL as bases de dados podem ser categorizadas como *key-value stores* [16] [17], base de dados de documentos [18] [19], base de dados de grafos ou implementações BigTable. As base de dados de grafos e implementações BigTable não foram consideras para a realização deste projecto.

B.1 Key-value stores

A função principal de um key-value store é guardar um valor associado a uma chave. Para essa função é disponibilizada uma variação da Api descrita na lista valuestore:

```
1 void Put(string key, byte[] data);
2 byte[] Get(string key);
3 void Remove(string key);
```

Lista 4: Api simplificada de um key-value store.

O valor guardado é um blob. Esta característica torna desnecessária a definição de um *schema* dando assim total flexibilidade no armazenamento de dados. Devido ao acesso ser feito através de uma chave este tipo de persistência pode facilmente ser optimizada e ter a sua performance melhorada.

- ***Schema*** - O *schema* neste tipo de base de dados é simples, a chave é uma string e o valor é um blob. O tipo e a forma como os dados são estruturados é da responsabilidade do utilizador.
- **Concorrência** - Num key-value a concorrência apenas se manifesta quando são feitas operações sobre a mesma chave.
- **Queries** - Uma *query* é apenas possível fazer com base na chave e como retorno obtêm-se o valor associado. Esta é uma limitação deste tipo de solução que não se manifesta em ambientes em que o único tipo de acesso necessário é com base numa chave, como é o case de um sistema de cache.

[5]BASE - **B**asically **A**vailable, **S**oft state, **E**ventual consistency.

- **Escalabilidade** - Existem duas formas para o fazer sendo que a mais simples seria separar as chaves. Separar chaves implica decidir a regra de separação, que pode separar as chaves com base no seu primeiro caracter e cada caracter é alojado numa máquina diferente, esta forma torna-se uma não opção quando a máquina onde está a chave não está disponível. Para resolver esse problema é usada replicação.
- **Transações** - A garantia de que as escritas são feitas no contexto de uma transação só é dada se for escrita apenas uma chave. É possível oferecer essas garantias para múltiplas chaves mas tendo em conta que um key-value store permite que diferentes chaves estejam armazenadas em diferenter máquinas torna o processo de dificil implementação.

B.2 Base de dados de documentos

Uma base de dados de documentos é na sua essência um key-value store. A diferença é que, numa base de dados de documentos, o blob de informação é persistido de uma forma semi-estruturada, em documentos, utilizando um formato que possa ser interpretado pela base de dados como JSON, BSON, XML, etc, permitindo realizar queries sobre essa informação.

- ***Schema*** - Este tipo de base de dados não necessita que lhe seja definido um *schema à priori* e não têm tabelas, colunas, tuplos ou relações. Uma base de dados orientada a documentos é composta por vários documentos auto-descritivos, ou seja, a informação relativa a um documento está guardada dentro deste. Isso permite que sejam armazenados objectos complexos (i.e. grafos, dicionários, listas, etc) com facilidade. Esta característica implica que, apesar de poderem existir referências entre documentos a base de dados não garante a integridade dessa relação.
- **Concorrência** - Existem várias abordagens para resolver este problema como a concorrência optimista, pessimista ou *merge*.
  - Concorrência Optimista: Antes de gravar informação é verificado se o documento foi alterado por outra transacção, sendo a transacção abortada nesse caso;
  - Concorrência Péssimista: Usa locks para impedir várias transacções de modificarem o mesmo documento. Esta abordagem é um problema para a escalabilidade destes sistemas;
  - Concorrência *merge*: Semelhante à concorrência optimista mas em vez de abortar a transacção permite ao utilizador resolver o conflito entre as versões do documento.
- **Transações**- Em alguns casos é dada a garantia de que as operações cumprem com a regra ACID (atomicity, consistency, isolation, durability). Algumas implementações optam por não seguir a regra ACID, desprezando algumas propriedades em detrimento de um aumento de rendimento, usando as regras CAP (Consistency, Availability, Partition Tolerance) ou BASE (Basically Available, Soft State, Eventually Consistent).

- ***Queries*** - As *queries* são feitas com base em índices inferidos automaticamente ou definidos explicitamente pelo programador. Quando o índice é definido o SGBD executa-o e prepara os resultados minimizando o esforço computacional necessário para responder a uma *query*.

  A forma de actualização destes índices difere em cada implementação deste tipo de base de dados, podendo ser actualizados quando os documentos associados a estes índices são alterados ou no momento anterior à execução da *query*. No primeiro caso isso significa que podem ser obtidos resultados desactualizados, uma vez que as *queries* aos índices têm resultados imediatos e a actualização pode não estar concluída. Na segunda abordagem, se existirem muitas alterações para fazer a *query* pode demorar algum tempo a responder.

- **Escalabilidade** - Este tipo tipo de base de dados suporta Sharding, ou seja partição horizontal, o que permite separar documentos por vários servidores.

B.3 RavenDB

O RavenDB [10] é uma base de dados de documentos implementada na *framework* .NET [7] que suporta a componente Linq [9] para *querying*. A base de dados é dividida em dois blocos o servidor e o cliente. O servidor é transaccional, armazena os dados no formato JSON e tem como interface um serviço web disponibilizado através do protocolo HTTP. O cliente tem como função expor todas as funcionalidades do servidor através de uma Api.

B.3.1 Cliente RavenDB

Para interacção com o cliente são usadas classes *POCO* (*Plain Old CLR Object*) o que torna desnecessária a utilização de um *ORM* ou qualquer sistema de correspondência entre objectos de domínio e os objectos persistidos. O cliente para além de gerir a comunicação com o servidor o cliente é responsável por fazer cache dos pedidos ao servidor e pela implementação do padrão *Unit of Work*.

A infra-estrutura utiliza principalmente duas classes do cliente, `IDocumentStore` e `IDocumentSession`. A classe `IDocumentSession` representa uma sessão e permite obter dados, persistir dados e apagar dados da base de dados. O padrão *Unit of Work* é implementado nas instâmcias desta classe e é dada a garantia que todas as alterações serão persistidas numa única transacção. A classe `IDocumentStore` é uma fabrica para a criação de sessões.

B.3.2 Relações entre documentos

As relações entre documentos, devido à inexistência de operações *JOIN*, podem ser representadas de várias formas. As formas consideradas são a utilização do identificador para incluir as entidades no pedido da entidade principal, a desnormalização e as *live projections*.

Inclusão no pedido Nesta opção cada documento guarda o identificador do(s) documento(s) com que está relacionado (e.g. os documentos que representam projectos guardam o identificador dos documentos que representam tarefas) e quando é obtido um documento principal são também obtidos todos os documentos que lhe estão associadas.

No RavenDB esta opção é suportada pelo método `Include` que recebe os identificadores das entidades a carregar em paralelo com as entidades principais. As entidades incluídas são adicionadas à sessão pelo cliente RavenDB.

Desnormalização A desnormalização de um documento consiste em extrair as informações relevantes, de outros documentos, e replicá-las no documento a persistir.

A figura **??** mostra um exemplo de desnormalização. O documento Projecto guarda para além do identificar dos utilizadores que lhe estão associados o seu Nome. Numa situação em que seja necessário apenas o Nome dos utilizadores associados essa informação já está no documento.

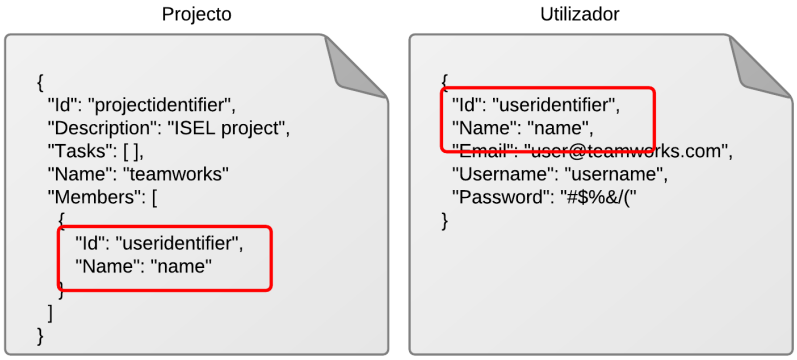

Figura 10: Exemplo de desnormalização de informação

Esta situação tem a vantagem de reduzir o número de pedidos à base de dados porque o documento guarda toda a informação necessária. Esta abordagem tem a desvantagem de que qualquer alteração a um documento desnormalizado implica a alteração de todos os documentos que o utilizam.

Live Projections O *RavenDB* oferece ainda forma de juntar e transformar documentos no servidor obtendo como resultado objectos diferentes dos objectos persistidos. Esta funcionalidade permite carregar documentos relacionados, escolhendo as propriedades de cada um que se pretende.

Na implementação deste projecto optou-se por estabelecer relações entre documentos através do seu identificador, tirando partido da funcionalidade *Include* oferecida pelo cliente RavenDB.

B.3.3 Bundles

No caso de as funcionalidades disponibilizadas pelo servidor RavenDB não serem suficientes existem Bundles que extendem as funcionalidades oferecidas. Os Bundles oferecidos com a build do RavenDB são:

- **Sharding and Replication**.
- **Quotas**, coloca limites ao tamanho na base de dados.
- **Expiration**, remove documentos expirados automáticamente.
- **Index Replication**, replica indices RavenDB para base de dados SQL Server.
- **Authentication**, autentica utilizadores na base de dados usando OAuth.
- **Authorization**, permite a gestão de grupos, *roles* e permissões.
- **Versioning**, automaticamente gera versões dos documentos quando são alterados ou removidos.
- **Cascade Deletes**, automatiza operações de remoção *cascade*.
- **More Like This**, retorna documentos relacionados com o documento indicado.
- **Unique Constraints**, adiciona a possibilidade de definir *unique constraints* em documentos RavenDB.

Os bundles disponibilizados são distribuidos com dois *dll*s, um para utilizar no cliente e outro para colocar numa pasta definida na configuração do servidor onde são colocadas todas as suas extensões.