Back end relatorio

Escolha de DB:

Devido ao approach tomado desde o inicio do desenvolvimento, front to back, não se pode falar desta escolha sem levar em conta a interface visual, com as seguintes caracteristicas principais:

1 - Todas as paginas principais são altamente relacionais, no sentido em que, na maioria das entidades, temos presentes informações de outras.

2 - A procura customizada de entidades, muitas vezes envolve caracteristicas compostas
Ex: somatorio de golos, golos por jogo (de jogadores e equipas)

3 - Conteudo dinamico em que a sua escala é pouco previsivel por ser uma rede social.

4 - A escolha de Angular, uma single-page application Framework.

Face á emergencia de base de dados NoSQL, criadas com redes sociais e outros casos de uso Web 2.0, optamos pela escolha de MongoDB, uma das DB's NoSQL mais maturas no mercado.

Como funciona MongoDB?

Como uma Base de dados orientada a Documentos, utiliza o conceito de dados e documentos auto-contidos e auto-descritivos, e isso implica que o documento em si já define como ele deve ser apresentado e qual é o significado dos dados armazenados na sua estrutura.

Ou seja, idealmente, por pagina irá apenas ser preciso 1 documento.

Com este conceito em mente, começamos por atacar as nossas principais características:

1 - Devido à informação relativa às restantes entidades ser, na sua maioria, dificilmente mutável (não se muda de nome todos os dias) e em pouca quantidade, a sua replicação não é, de todo, critica.

2 - As características compostas como somatórios e agregações estão auto contidas nos documentos, sendo a sua procura mais rápida, embora mais custosa em termos de actualização de base de dados

3 - O ponto mais preocupante desta escolha no ponto de vista teórico, isto porque um documento pode ter a o tamanho máximo de 14MB, no ponto de vista prático isto não é tão realista, pois metendo em perspectiva:
O aclamado livro “A Guerra dos mundos “ por H. G. Wells de 287 páginas, tem apenas 400KB, e mesmo que uma bibliografia de um jogador seja superior, há métodos de divisão de documentos disponíveis.

4 - Quanto menos pedidos forem feitos á base de dados, mais responsive será uma single page application, por termos apenas que pedir 1 ou 2 documentos à base de dados invés de vários pedidos por informação relacional como em base de dados SQL, a nossa aplicação apresenta load times muito mais curtos.

Argumentos relativamente à optimização de queries e escalabilidade da base de dados:

-> O nosso conteúdo relacional é pouco clusterizado
Ex: os golos, assistências e cartões de um jogador,durante uma época, não ficam armazenados em disco relativamente perto uns dos outros, num contexto de base de dados SQL.

Em MongoDB, todo esse conteúdo fica num único documento, evitando Fetch’s a várias secções do disco.

-> MongoDB tem um sistema de “sharding” que permite, com facilidade, uma escalabiilidade horizontal da base de dados.

-> Posso pensar mais um bocado, mas acho que já está too much.

ESTRUTURA

(mete a imagem que tinhas da outra vez)

(mete a imagem do esquema que temos agora (mais simples, por falta de tempo))

Esta parte tu ja descreves-te bem no teu relatorio anterior, já nao o tenho aqui, esta parte fazemos os 2 em conjunto.

CRAWLERS

Escolha de framework

O processo de crawling está intimamente ligado á logica de negócio, estamos a falar de centenas de equipas e milhares de jogadores, com potencial de escalar varios niveis de grandeza no futuro, logo a framework tem que ser robusta e versatil.

A framework que escolhemos usar chama-se crawler-js:

-> Funciona a partir de uma fila de pedidos com varios workers, esta fila é altamente configuravel, podendo-se configurar proxies, tempos maximos entre pedidos, entre outros.

-> Pode-se usar a interface de JQuery para navegar pelo DOM, algo bastante util devido a nossa experiencia tecnologica com a mesma.

-> Open Source e sem custos.

Hierarquia de Entidades e Paralelismo

Devido à quantidade de páginas a ser crawled, optarmos por um processo que seja possivel paralelizar, seria um ideal, embora isto implique abordarmos vários problemas:

-> Atomaticidade das operações.

Em MongoDB as operaçoes sao atomicas ao nivel do documento, inclusive o lock do mesmo, logo qualquer operação, desde que seja atómica e auto contida por si (ou seja, ausencia de SET's, tudo a base de incrementaçoes e outras operaçoes sem conhecer o estado corrente)

-> Dependencias entre os modelos e Concurrencia

Devido a Modularização dos modelos em si (nenhum destes preciso obrigatoriamente de outro para ter alguma identidade), a hierarquia é feita pelo quão contidas estão as entidades com a subida do nível hierárquico.

Competição:
- Contém poucas dezenas de equipas
- Contém centenas de jogadores
- Contém centenas de jogos

Equipa:
- Contem dezenas de jogadores
- Contem centenas de jogos mas de competiçoes diferentes

Jogo:
- Contem 2 equipas
- Dezenas de jogadores
- Pertence apenas a 1 competição.

Os jogos serão crawled por base das competições (todos os jogos de uma competiçao, ao inves de todos os jogos de uma equipa), para garantir que quando os mesmos estão a ser processados, não exista um constante lookup da existencia dos jogadores ou equipas do mesmo, pois estes já vão estar devidamente processados, para isto acontecer modificamos a estrutura da pilha da framework para uma priority queue em que nenhum pedido de uma prioridade inferior pode começar sem serem completos todos os pedidos de hierarquias anteriores.

Como estamos a fazer crawling de modelos numa base de dados externa que funciona por REST, podemos usar os ID's de todas as entidades como chaves da nossa procura, não sendo preciso reconhecer o ID na nossa base de dados, isto torna o processo de actualização dos modelos muito mais simples