Relatório de Progresso

PlayersNet

Disciplina: Projeto e Seminário
Curso: Engenharia Informática e de Computadores

Arnaldo Tema
Nº aluno: 39342
Lisboa, 30 de Abril de 2018

Índice

Progresso

Arquitetura Geral do Projeto

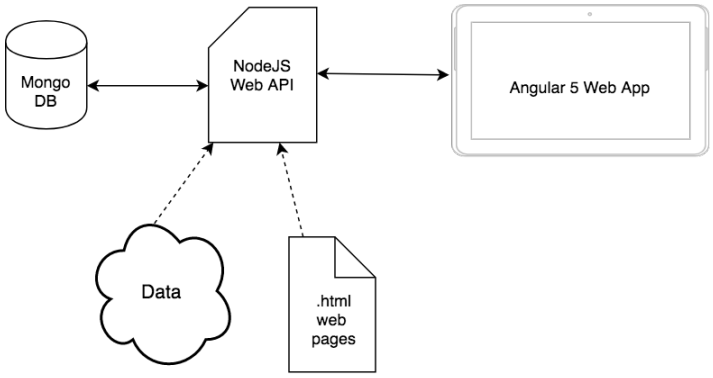


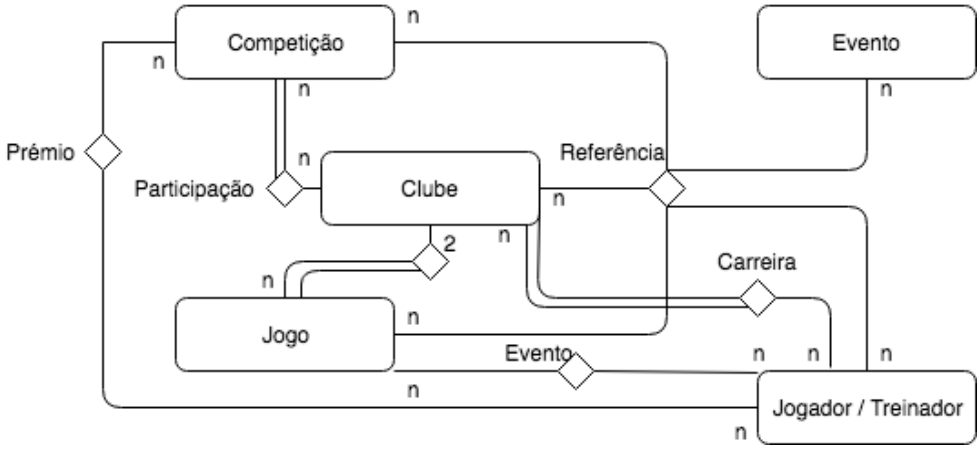
figura 1

Tal como a maioria dos projetos aplicacionais, a PlayersNet divide-te em 3 camadas – Base de dados em MongoDB, Servidor em NodeJS, Front-end em Angular 5.
Para além dos dados inseridos por parte dos utilizadores, o servidor está preparado para obter também informação de outras fontes (outras bases de dados ou outros sites).

Base de dados

Primeira Abordagem

Antes de se ter optado por utilizar uma base de dados não relacional, tinha sido definido o seguinte modelo EA

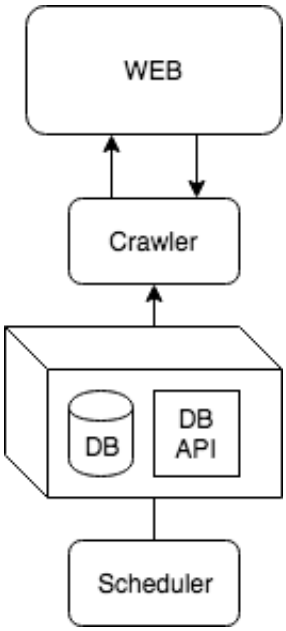


Abordagem corrente

Após uma análise mais cautelosa, chegou-se à conclusão que uma abordagem com um sistema de base de dados não relacional permitiria uma base de dados mais versátil e de fácil indexação – consequentemente, de melhor e mais rápida pesquisa. Assim, passam a existir os documentos (modelos) presentes na figura ao lado, que se interligam através de referências simples (mensagens, recomendações, notificações, entre outros) que ainda não se encontram definidas.

Arquitetura

Primeira Abordagem



Web
Componente Front-End da aplicação que acarreta apenas a *user interface* do projeto. Todas as alterações de configuração da aplicação têm de ser feitas manualmente. Esta componente comunica com a componente Crawler.

Crawler
Componente responsável por recolher dados automaticamente de outros sites e mapeá-los na base de dados. Esta componente é também responsável por todo o processamento de Back-End da aplicação, tais como: autenticação, autorização, mensagens, envio automático de emails e comunicação com a componente de armazenamento de dados. A componente Scheduler é que determina os timings em que o crawler irá percorrer as páginas que contêm informação.

DB Api
Componente que trata de toda a comunicação com a base de dados. Esta componente recebe os valores que o crawler envia (quer sejam os valores automáticos, quer os que vêm da componente web).
(Nota, no diagrama 1 falta a seta que traduz a informação que esta componente recebe por parte do crawler)

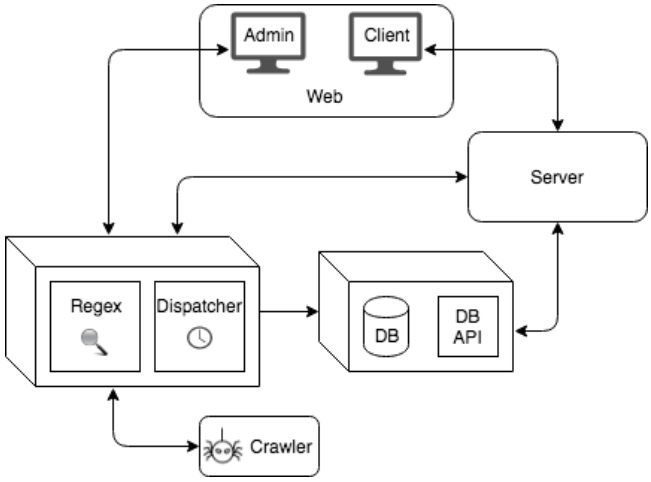
DB
Base de Dados MongoDB.

Scheduler
Componente de configuração do crawler. Aqui situa-se o código responsável por lançar eventos de começo e fim do crawler, trata do sistema de Logs do que foi obtido pelo crawler e guarda-los em ficheiros de texto para efeitos de análise.
(Nota, no diagrama 1 falta a seta que traduz a informação que esta componente recebe por parte do crawler)

Abordagem corrente

Web

Componente Front-End da aplicação que, não só acarreta a *user interface* do projeto (client), mas também páginas que permitem a configuração específica de alguns eventos (admin).



Desta forma, alterações de configuração tais como:
- O intervalo de tempo em que corre o crawler (comunicação com o *dispatcher*),
- Critérios de prémios dos jogadores (comunicação com o *regex*),
- Critérios de atribuição de badgers a jogadores ou treinadores (comunicação com o *regex*)

passam a poder ser feitas por um utilizador administrador.

Dispatcher

Componente responsável por lançar eventos automáticos que o *crawler* irá consumir. Para além disso, é no *dispatcher* que se encontram as ações que determinam as configurações definidas pela componente *admin* – referidas na componente anterior e é nesta componente que é são escritos na base de dados os valores obtidos pelo crawler. O *dispatcher* está também encarregue por lançar eventos detetados na informação obtida pelos crawlers (como por exemplo, um jogador marcou um hat-trick, ou é a décima vitória consecutiva do clube XPTO). A deteção da maioria desses eventos é definida pela componente *regex*.

Regex

Nesta componente encontra-se toda a lógica que determina as condições necessárias para que um evento seja despoletado.
De forma a reduzir a quantidade de condições necessária para que se capture os inúmeros eventos pretendidos (tais como 3 golos marcados num jogo, 5º jogo consecutivo sem marcar golos, 34º jogo consecutivo sem perder, etc.) optou-se por atribuir a cada entidade (clube, jogador ou treinador) um texto codificado que identifica diferentes eventos, por exemplo, “G” representa um golo, “W” representa uma vitória e “L” representa uma derrota.
Assim, através de um sistema de regular expressions que vai percorrer os enormes textos dos utilizadores em questão, obtêm-se as respostas pretendidas e consegue-se detetar a ocorrência dos eventos definidos.

Crawler

Componente responsável por recolher dados automaticamente de outros sites e retorná-los ao *dispatcher*. A única informação que o *crawler* recebe são dados de configuração e estes vêm por parte do *dispatcher*.

DB Api

Componente que trata de toda a comunicação com a base de dados. Esta componente recebe os valores que o *dispatcher* recebe do crawler, os valores que o *dispatcher* recebe dos administradores, envia (quer sejam os valores automáticos, quer os que vêm da componente web).

DB

Base de Dados MongoDB.

Server

Componente servidora principal. Contém todos os endpoints da aplicação (Web Api) e é responsável por todo

o processamento de Back-End da aplicação, tais como: autenticação, autorização, mensagens, envio automático de emails e comunicação com a componente de armazenamento de dados.

Web Crawler

Foi desenvolvido em NodeJS uma API que percorre o site *www.zerozero.pt* de forma a obter o máximo de dados de todos os jogadores, treinadores, clubes de futebol, resultados e classificações das primeiras divisões dos escalões sénior e últimos de formação do país.

A API utiliza a biblioteca *Crawler-js* para ler os valores das páginas HTML do site acima referido, os valores são em seguida mapeados para os documentos (modelos) existentes na nossa base de dados em Mongo, através da biblioteca *Mongoose*.

Para além de permitir assim uma sólida quantidade de informação dos utilizadores principais na plataforma independente de insumos dos mesmos, juntamente com o *dispatcher* - que determina os períodos de "fetch" de informação - é possível obter dados cuja necessidade de que estejam o mais atuais possível sejam constantemente atualizados em curtos períodos de tempo. Tais como resultados de jogos, estatística, entre outros.

(Esta componente encontra-se em fase de teste)

Interface de Utilizador

Na camada de apresentação, apesar de se tratar de desenvolvimento front-end, como está a ser desenvolvida em Angular 5 existe um padrão de desenho bem explícito na sua estrutura.

Sendo que esta camada se pode dividir em "controladores", "modelos" e "vistas", a estrutura Angular é muitas vezes confundida com um padrão de desenho MVC ou MVVM. No entanto, não há por defeito controladores ou *ViewModels* no Angular realmente, há sim componentes que derivam de *templates*, serviços e modelos.

Posto isto, na camada de front-end tem-se uma subcamada de modelos que representam os que se encontram na base de dados, outra de serviços onde se encontram os pedidos ao servidor que mapeiam os modelos e também outros serviços como os de lógica de autenticação e autorização, e, por fim, a subcamada visual que, com o Angular, nos permite dividir por cada componente diferente (i.e. player-component, team-component, header-component, footer-component, chat-component, skills-component) especificando um ficheiro .html, .css e .ts (este último funciona como a classe da componente) por cada um deles tornando possível ter características bastante específicas por componente.

Olhando para a estrutura completa desta camada, pode-se notar uma orientação a objetos subtil na forma como a mesma é implementada.

À data corrente já se encontram definidas as subcamadas models, services e a subcamada visual, e parcialmente implementados os componentes player e team e respetivos serviços.

Problemas

Alteração da arquitetura

Como consta no capítulo que explica a arquitetura do projeto, houve uma significativa alteração do que tinha sido inicialmente definido que iria ser a estrutura da aplicação.

A primeira abordagem encontrava-se pouco dividida, o que acrescia o número de dependências entre componentes, tornava a aplicação pouco suscetível a alterações futuras que, num projeto desta categoria, tende a acontecer à medida que se vai desenvolvendo e havia componentes (como o *crawler*) que eram responsáveis por vários tipos de processamento.

Na abordagem corrente já se pode encontrar uma arquitetura mais segmentada, bastante sensível a alterações e melhorias sem que interfira com a lógica funcional do projeto, com componentes com propósitos bem definidos e de melhor compreensão aos olhos de quem tenta interpretar a lógica do projeto.

No entanto, esta alteração, bem como a definição dos modelos da base de dados não relacional (MongoDB funciona apenas por referências entre entidades), foram causa de uma demora maior nesta fase de desenvolvimentos, o que trará consequências nos tempos estimados para outros desenvolvimentos.

Dependência dos Sites para os Crawlers

Tal como se verifica nas explicações do funcionamento do *web cralwer* desta aplicação, há uma **enorme** dependência das fontes do mesmo para que o funcionamento saia ileso e com sucesso. Ou seja, basta que uma das páginas lidas pelo *crawler* seja alterada para que o mesmo deixe de funcionar corretamente.

Para solucionar este problema, foi necessário fazer uma análise da probabilidade da alteração das páginas em questão junto com os administradores do site das mesmas da qual se concluiu que, apesar de muito remota, existe obviamente essa possibilidade. Portanto, terá de ser criado no *dispatcher* um sistema de Logging que notifique essas alterações para que rapidamente se possa atualizar o crawler em função das mesmas. Não sendo esta uma definitiva, o objetivo é que, futuramente, se consiga obter todos os dados diretamente da mesma fonte onde o site de onde, neste momento, se está a extrair informação a consegue obter.

Tempo para o desenvolvimento Front-End

Seguindo a explicação no primeiro problema apontado neste capítulo, o tempo escasseia-se para alguns desenvolvimentos que ainda restam acontecer.

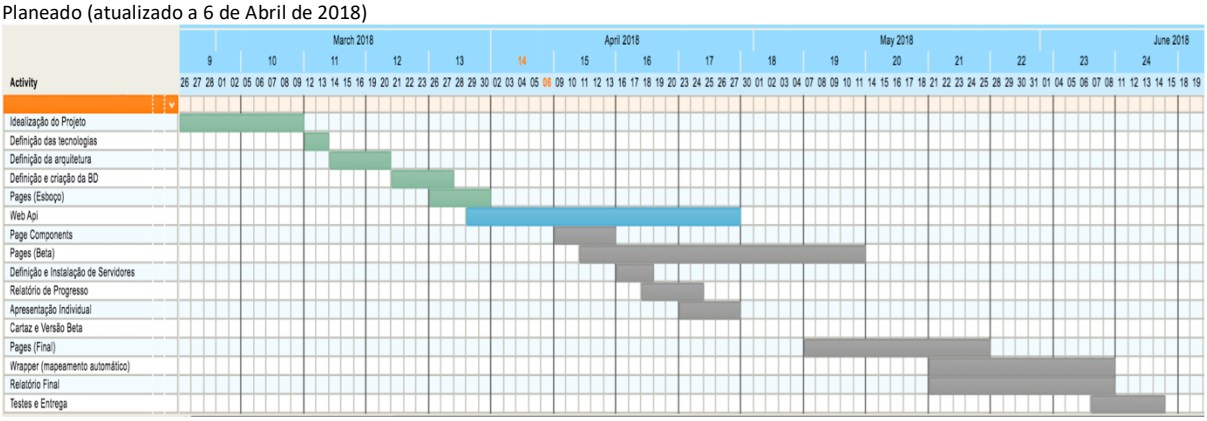
Visto que todo o projeto carece de um workflow altamente elaborado, os desenvolvimentos de back-end não poderão, de forma alguma, sair prejudicados, pondo em causa o funcionamento geral da aplicação. Logo, se realmente houver necessidade de abdicar de alguns desenvolvimentos, será a camada de apresentação a sacrificada.

Apontando sempre para uma apresentação perfeitamente funcional e apresentável, a versão final deste projeto poderá ficar apenas menos “bonita” do que uma eventual versão futura onde haja uma maior alocação de horas de desenvolvimento para a componente visual.

Plano

Segue o plano de trabalho planeado e corrente.

Planeado (atualizado a 6 de Abril de 2018)



Estado corrente (25 Abril de 2018)