FRAMEWORK ORIENTADO A OBJETOS PARA ANÁLISE ESTATÍSTICA DE BASQUETE

Isaac Nunes Borges, Luciana Pereira de Araújo Kohler – Orientadora

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brazil

isaac.borges@furb.br, lpa@furb.br

Resumo: *Este trabalho apresenta o desenvolvimento de um framework para análise estatística de basquete, comparando e analisando as características de equipes e jogadores. O framework desenvolvido em Visual Studio com a linguagem C# tem por objetivo disponibilizar os principais métodos através de interfaces concretas, realizando toda a persistência das informações no banco de dados não relacional Mongo DB, de modo a abstrair essas informações das aplicações que a utilizarão. O trabalho apresenta o desenvolvimento deste framework e também de uma aplicação web utilizando os recursos deste framework. As vantagens de se utilizar o framework são um menor esforço na construção da aplicação para análise de estatísticas do basquete, além de seguir um padrão de desenvolvimento utilizando a herança de classes implementadas no framework. O resultado foi uma aplicação web com interface utilizando gráficos e tabelas para o usuário utilizando as implementações do framework.*

Palavras-chave*: Framework. Basquete. Análise estatística.*

1 INTRODUÇÃO

Na União Europeia, o esporte é mais importante para a economia do que a agricultura, pois representam 1,13% e 1,76% do Produto Interno Bruto (PIB) respectivamente (CALEIRO, 2014). Caleiro (2014) complementa destacando a importância da indústria esportiva no Brasil, sendo que o PIB do esporte cresceu a taxas anuais de 7,1% entre 2007 e 2011. A avaliação é que o esporte respondia em 2012 por 1,6% do PIB do país, equivalente a R$ 67 bilhões (CALEIRO, 2014). O mercado esportivo mundial projetou para 2017 receitas de US$ 90 bilhões, representando um aumento de quase 100% nos últimos 12 anos (MAGNUS, 2018). Magnus (2018) explica que os números não são por acaso, o uso da mineração de dados nesse segmento cresce na mesma proporção (o setor de análise esportiva deve crescer de modestos US$ 125 milhões em 2014 para incríveis 4,7 bilhões em 2021).

Segundo Ferreira (2017), empresas que investem em soluções de Big Data estão naturalmente na vanguarda do mundo de negócios. Pode-se destacar alguns exemplos de como o Big Data ou a análise em tempo real pode dar margem para empresas competitivas, ou reduzir os gastos e prejuízos. Olhando para o mercado financeiro, nos quais existem grandes transações por segundo, é necessário além do banco de dados, o uso de algoritmos sofisticados para processar o alto volume de dados em tempo real (IANNI, 2016). A indústria esportiva obteve benefícios com o crescimento da análise estatística ou análise de volume de dados. Os dados de desempenho coletados em tempo real, como frequência cardíaca, altura do impulso ou queda de rendimento muscular, permitem identificar com maior precisão e agilidade os limites físicos dos atletas (HEKIMA, 2015).

O basquete é um dos esportes que abraçou o Big Data e é um exemplo para outras modalidades em busca da modernização e utilização dessas informações ao seu favor (KATCHBORIAN, 2016). Hekima (2016) descreve como a National Basketball Association (NBA), liga de basquete americano, usa o Big Data para otimizar a experiência do cliente. Segundo Hekima (2016) através da coleta, análise e visualização dos dados em tempo real, a experiência dos torcedores pode sair do contexto da quadra, uma vez que inúmeras estatísticas e outros recursos são disponibilizados para o público sob demanda. Diante do cenário apresentado, este trabalho apresenta o desenvolvimento de um *framework* para realizar a leitura de uma grande massa de dados informada pelo usuário, e converte-los em dados estatísticos.

Dessa forma o objetivo desse trabalho é fornecer um *framework* orientado a objetos para a análise estatística sobre o basquete que auxilie na identificação de qualidades e características de jogadores e equipes. Já os objetivos específicos são: (a) disponibilizar métodos para identificar as características das equipes e jogadores a cada temporada, filtrando por um critério a ser escolhido; (b) fornecer uma função para comparar as estatísticas da equipe a ser definida comparando com as outras equipes ao longo da história; (c) fornecer uma função de comparação entre um jogador a ser definido com outros jogadores da mesma posição analisando suas estatísticas; (d) desenvolvimento de uma aplicação utilizando o *framework*.

O artigo segue dividido da seguinte forma. A seção 2 aborda sobre a fundamentação teórica deste trabalho. A seção 3 explica o desenvolvimento do *framework*, com análises de diagramas e exemplos de código fonte. A seção 4

apresenta os resultados obtidos com o *framework*, demonstrado uma aplicação construída utilizando algumas funções do *framework*. Por fim a seção 5 é apresentado as conclusões do trabalho e as sugestões de extensão para este.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção está dividida em cinco subseções. A subseção 2.1 apresenta as definições e conceitos do basquete e suas posições. A subseção 2.2 descreve os conceitos da análise estatística e como a análise estatística está presente no esporte. A subseção 2.3 apresenta a aplicação das estatísticas nos esportes. A subseção 2.3 descreve os conceitos de *framework*. Por fim, a subseção 2.4 descreve os trabalhos correlatos

2.1 DEFINIÇÕES E CONCEITOS DO BASQUETE

Vieira e Freitas (2006, p. 6) afirmam que "Um jogo de basquete prevê a disputa em quadra entre duas equipes, compostas por 5 jogadores cada. O time que, no tempo de jogo, fizer mais pontos no campo adversário sai vitorioso". O basquete, assim como outros esportes, possui movimentos com mais frequência em determinadas situações do jogo. Esses movimentos são chamados de fundamentos. Rose Júnior e Tricoli (2005, p.7) definem fundamentos como os "gestos básicos do jogo, que podem ser executados isoladamente ou combinados com outros fundamentos, mas que dependem das capacidades motoras e coordenativas do atleta.".

No basquete, os jogadores são distribuídos na quadra em cinco posições distintas, sendo que cada posição possui características que a diferencia das demais. O Quadro 1 elaborado pela USA Basketball (2015) aborda o conceito de cada uma destas posições.

Quadro 1 - Posições do basquete

Posição	**Definição**	**Características**
Armador	O armador executa o ataque. Seu papel é importante para a equipe por ser uma extensão do treinador na quadra.	Conhecimento do jogo, bom controle de bola e passe, possuir velocidade e habilidade para ler as defesas.
Ala-armador	Geralmente o melhor pontuador dos armadores. O ala-armador tem a função de ajudar o armador na criação das jogadas.	Habilidade de infiltrar na defesa e distribuir a bola, bom controle de bola, velocidade, capacidade de acertar arremessos de longa distância.
Ala	Geralmente o jogador mais talentoso da equipe, deve ser um bom pontuador e capaz de marcar os cantos da quadra.	Rápido e veloz, capaz de pegar rebotes, bom passe, capaz de correr nos contra-ataques.
Ala-pivô	O ala-pivô é conhecido como o jogador que faz o "trabalho sujo" juntamente com o pivô controla as ações no garrafão.	Deve possuir uma sólida capacidade de rebote ofensivo e defensivo, habilidades de passe e arremesso.
Pivô	O pivô geralmente é o jogador mais alto em quadra, deve ser o líder defensivo e dominar o garrafão.	Agressividade nos rebotes e tocos, um pontuador confiável de curta distância, desenvolver um arremesso, capacidade de impulsão.

Fonte: Basketball (2015).

Essas definições são importantes para o trabalho, uma vez que os cálculos e pesquisas históricas dos jogadores utilizam a separação das categorias por posição.

2.2 ANÁLISE ESTATÍSTICA NO ESPORTE

A estatística pode ser definida como a Ciência que tem por objetivo a coleta, análise e interpretação de dados qualitativos e quantitativos (FAVERO, 2017). Favero (2017) complementa a definição de estatística como um conjunto de métodos para coleta e organização, resumo, análise e interpretação de dados para tomada de decisões. Rosa (2012) define a análise estatística como o processo de organização, processamento, e retirada de conclusões sobre um determinado conjunto de dados. Gregori (2015) acrescenta que se pode classificar a análise de estatísticas em dois tipos: descritivas e conclusivas. Segundo o autor, as análises descritivas são utilizadas para que organizações possam resumir os dados do mercado em que atuam, enquanto as análises conclusivas são elaboradas para estudar os dados coletados de uma maneira mais objetiva, pois partem de perguntas muito específicas que deverão ser respondidas através do processo analítico.

O termo Big Data leva como base os 3 Vs, que são: volume, velocidade e variedade. O volume está relacionado com a grande quantidade de dados que se tem em mãos, a velocidade baseia-se em que a cada segundo na internet são criados novos dados que podem ser uteis para a organização e por fim, a variedade de dados que pode ser um texto compartilhado em uma rede social, como um *review* em um site de compras (TARIFA, 2014). Favero (2017), no entanto, entende a existência de cinco características com respeito à geração de dados: volume, velocidade, variedade, variabilidade e complexidade. A combinação dessas cinco características de geração e disponibilidade recebe o nome de Big Data. Dentro desse contexto, as organizações dos mais diversos setores têm levado a investirem no

desenvolvimento da área Business Analytics, que possui o objetivo de analisar e gerar informações, permitindo estar a frente ao mercado e aos competidores (FAVERO 2017).

Uma vasta quantidade de dados está presente em todas as áreas do esporte. Estes dados representam qualidades e características individuais de cada jogador ou mesmo eventos que acontecem durante a partida como, por exemplo, como uma equipe está funcionando, quais características do plano de jogo estão sendo seguidos. (IVANKOVIC et al., 2010). De acordo com Ivankovic et al. (2010), existem diferentes níveis de abordagem entre as associações esportivas e a análise de dados, sendo:

a) sem conexão entre dados esportivos e sua utilização;
b) previsões baseadas em palpites e instintos de especialistas;
c) previsões baseadas em dados coletados;
d) utilização de estatísticas no processo de decisão;
e) uso da mineração de dados no processo de tomada de decisão.

A maioria das grandes ligas de basquete utilizam o terceiro ou quarto método de abordagem (IVANKOVIC et al., 2010). Embora o uso da análise esportiva tenha sido introduzido no esporte recentemente, os resultados obtidos por equipes que fazem uso desta tecnologia são excelentes (IVANKOVIC et al., 2010). Conforme levantamento realizado por Maxcy e Drayer (2014), é possível notar que o uso das estatísticas é tendência entre a maioria das equipes das grandes ligas americanas. Maxcy e Drayer (2014) apontam o uso da análise estatística em 56% na liga de futebol americano, 80% na liga de basquete e chegando a 97% de utilização na liga de baseball. Um exemplo da análise esportiva aplicada é na liga de basquete norte-americana, chamada de NBA, que possui 30 times e ao menos metade destes tem influentes diretores de estatísticas que possuem poder de decisão na hora de contratar jogadores ou mesmo na mudança de escalação durante os jogos (COBOS, 2010).

2.3 FRAMEWORK

Segundo Fayad, Schmidt e Jonhson (1999), *framework* é uma aplicação reusável e semi completa que pode ser especializada para produzir aplicações personalizadas. Ao contrário das técnicas usuais de orientação a objetos que organiza classes em bibliotecas, *frameworks* são voltados para unidades particulares de negócios e domínios de aplicação. Os principais benefícios em desenvolver um *framework* são a modularidade, reusabilidade, extensibilidade e inversão de controle.

Segundo Wirfs-Brock (1991) apud Silva (2000), pode-se definir *framework* como "um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo *framework*". Os *frameworks* Orientados a Objetos (OO) constituem uma abordagem de desenvolvimento de software que visa maximizar a reutilização de código. Os *frameworks* OO definem o fluxo de controle dos artefatos que serão produzidos a partir deles. Isto é, o projeto de estrutura OO consegue prever o que precisa ser feito pelos desenvolvedores em termos de quais classes devem ser especializadas e quais métodos substituídos. No caso ideal, seguindo esse princípio, todas as classes produzidas para completar a estrutura de orientação a objetos devem ser especializações de uma estrutura OO (SILVA e FREIBERGER, 2008).

De acordo com Wirfs-Brock (1990, apud Silva 2000), o *framework* é uma coleção de classes concretas e abstratas e as interfaces entre eles é o projeto de um subsistema. Dois aspectos caracterizam um *framework*: os *frameworks* fornecem infraestrutura e projeto e os *frameworks* "chamam" e não são "chamados" (TALIGENT 1995 apud SILVA 2000). A obtenção de boa estrutura para implementação, os recursos diversos que proporcionam a facilidade para construir um projeto de software e o cumprimento pleno da documentação, ajuda a diminuir custos estabelecidos e aumenta a produtividade dos desenvolvedores em relação à construção do projeto de software (SOMMERVILLE, 2008).

2.4 TRABALHOS CORRELATOS

Esta seção apresenta alguns trabalhos e ferramentas comercias com finalidades semelhantes ao apresentado neste trabalho. Entre os trabalhos selecionados estão a análise de basquete utilizando redes neurais realizado por Ivankovic et al. (2010) descrito no

Quadro 2. O Quadro 3 apresenta o trabalho de Canan, Mendes e da Silva (2015), com uma análise estatística no basquetebol de base. O Quadro 4 descreve a ferramenta comercial de projeção de jogadores chamada CARMELO. Por fim o Quadro 5 apresenta o projeto de Cao (2012) para prever os resultados das partidas.

Quadro 2 - Analysis of Basketball games using neural networks

Referência	Ivankovic et al. (2010).
Objetivos	Identificar as estatísticas mais importantes no basquete, quais destas estatísticas mais afetam o resultado de uma partida.
Principais funcionalidades	Os autores utilizaram a mineração de dados para coletar estatísticas da liga B de basquete da Sérvia entre as temporadas de 2006 a 2010.
Ferramentas de desenvolvimento	Os dados foram analisados utilizando a técnica de feedforward em redes neurais como um método de mineração de dados. Ivankovic et al. (2010) utilizaram comandos SQL para fragmentar, selecionar e relacionar os dados para conseguirem encontrar os resultados.
Resultados e conclusões	A conclusão geral das análises realizadas por Ivankovic et al. (2010) foram que um jogo no garrafão é crucial para ganhar a partida, ofensivamente os arremessos de 2 pontos e defensivamente os rebotes são os aspectos mais importantes a serem considerados. Os autores obtiveram um total de 80,96% de acertos do time vencedor utilizando os dados coletados.

Fonte: elaborado pelo autor.

Quadro 3 - Análise estatística no basquetebol de base

Referência	Canan, Mendes e da Silva (2015).
Objetivos	Os autores buscaram respostas para questões como: quais as características quantitativas predominantes nos jogos de basquetebol no referido contexto? Quais os padrões de acontecimentos que definem equipes mais ou menos vitoriosas?
Principais funcionalidades	O método abordado pelos autores foi utilizar a análise estatística de todos os jogos que foram realizados pelo Campeonato Paranaense de Basquetebol Masculino Sub-17, no ano de 2011. Cada equipe disputou um total de 15 partidas, em cada uma delas foi realizado um trabalho de coleta, cálculo e análise de 22 indicadores estatísticos.
Ferramentas de desenvolvimento	Canan, Mendes e da Silva (2015) realizaram as análises *in loco*, sem gravação em vídeo. A coleta foi elaborada utilizando o programa Microsoft Office Excel 2007. Os indicadores coletados não apresentaram margem de erro comparados as estatísticas geradas pelo site do campeonato.
Resultados e conclusões	Baseando-se nos indicadores estatísticos, o resultado do jogo mostrava que a equipe vitoriosa apresentava médias superiores em 19 dos 22 indicadores coletados. Os indicadores relacionados a arremessos de dois pontos, porcentagem de arremessos, rebotes e assistências foram considerados mais significativos.

Fonte: elaborado pelo autor.

Quadro 4 - CARMELO NBA Player Projections

Referência	FiveThirtyEight (2017).
Objetivos	Identificar jogadores através da história da NBA para prever com quem o jogador em questão se assemelha.
Principais funcionalidades	Utilizando um modelo probabilístico de previsão é possível comparar as estatísticas do jogador no mesmo período da vida com outros atletas identificando suas similaridades.
Ferramentas de desenvolvimento	Ferramenta comercial
Resultados e conclusões	Ferramenta comercial

Fonte: elaborado pelo autor.

Quadro 5 - Sports Data Mining Technology Used in Basketball Outcome Prediction

Referência	Cao (2012).
Objetivos	Prever os resultados dos jogos da NBA
Principais funcionalidades	O projeto concentra-se no uso de algoritmos de aprendizado de máquina para construir um modelo para prever os resultados da NBA. Os algoritmos envolvem um classificador logístico e redes neurais. Cao (2012) utilizou estatísticas avançadas de 5 temporadas da NBA para criar a base de dados.
Ferramentas de desenvolvimento	O autor utilizou a linguagem de programação Ruby e um banco de dados MySQL hospedado na Amazon EC2.
Resultados e conclusões	Cao (2012) realizou a comparação dos resultados obtidos com diferentes fontes e pode notar uma melhor taxa de acerto de seu projeto, uma vez que a quantidade de dados processadas fora considerável. O trabalho conseguiu atingir uma precisão de 69,67%.

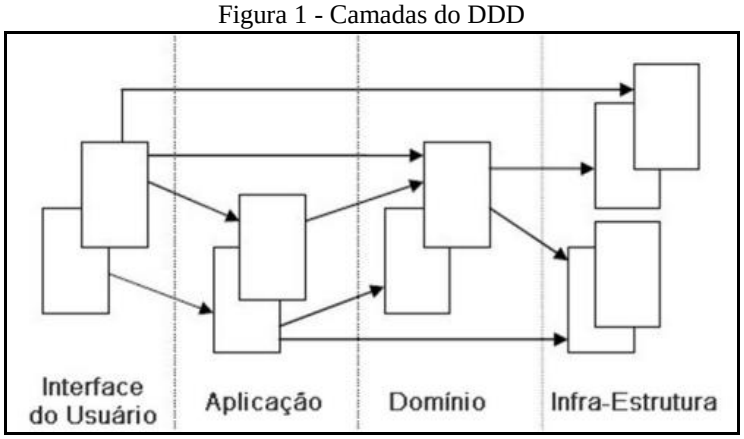
Fonte: elaborado pelo autor.

3 DESCRIÇÃO DO FRAMEWORK

Nesta seção serão descritos as especificações e detalhamento do *framework* desenvolvido, apresentando suas características, técnicas e ferramentas utilizadas. A seção também contempla os detalhes da aplicação desenvolvida para testar o *framework.*

O *framework* foi desenvolvido utilizando a ferramenta Microsoft Visual Studio na sua versão Enterprise 2017, sendo codificado na linguagem C#. A arquitetura do *framework* foi baseada no conceito de Domain Driven Design (DDD) com o foco na regra de negócio ou simplesmente domínio. Franco et al. (2010) descrevem o DDD como uma maneira de pensar e um conjunto de princípios que ajudam os desenvolvedores a criarem sistemas. Não se trata de uma tecnologia ou metodologia, mas sim de uma abordagem de projeto de software, focado no domínio e lógica do sistema. A Figura 1 mostra as camadas de uma solução seguindo os preceitos do DDD.

Figura 1 - Camadas do DDD



Fonte: Franco et al. (2010).

Conforme a Figura 1, para se aplicar o conceito do DDD é necessário separar o projeto em camadas, com o objetivo de segregar as funções e deixar cada implementação com suas responsabilidades. Aplicando esses conceitos é mais simples realizar a troca de tecnologia sem implicar na camada de domínio. O foco do DDD está na camada de domínio da arquitetura, responsável pelos conceitos de casos de uso e regras de negócio. Cada camada deve exercer o papel a qual se destina, sendo o isolamento da camada de domínio um pré-requisito para o DDD (FRANCO et al., 2010).

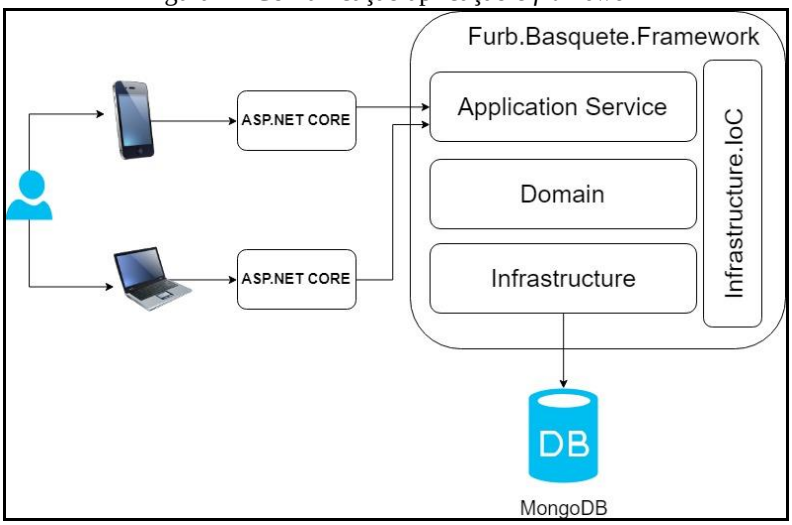
Para o gerenciamento dos dados foi utilizado o banco não relacional MongoDB. Além do *framework* foi desenvolvida uma aplicação para testá-lo. Dessa forma, essa aplicação responsável por utilizar o *framework* foi desenvolvida também no Microsoft Visual Studio. Foram implementadas páginas web para a interface com o usuário utilizando a tecnologia Razor com HTML 5 e o *framework* Bootstrap. Para apresentar os resultados estatísticos utilizou-se de gráficos que foram construídos com o auxílio dos componentes visuais da empresa Syncfusion.

O *framework* foi dividido em três partes, sendo elas:

a) Application Service: camada de comunicação entre as aplicações externas e o domínio, além de algumas funcionalidades mais específicas como importação dos dados;
b) Domain: camada responsável por realizar as regras de negócio, cálculos e pesquisas sem dependência com nenhum outro projeto;
c) Infrastructure: camada separada em dois projetos, o primeiro fica responsável por comunicar-se com o banco de dados e retornar as informações para a camada de domínio, enquanto o segundo realiza a Injeção de dependência.

Na Figura 2 pode-se identificar a comunicação de aplicações desenvolvidas com ASP.NET CORE consumindo recursos do *framework*. Essas aplicações podem ser apresentadas para o usuário no formato web ou de aplicativo. O *framework* fica responsável por gravar e consultar as informações no banco de dados, neste caso implementado utilizando um banco de dados não relacional.

Figura 2 - Comunicação aplicação e *framework*



Fonte: Elaborado pelo autor.

O Quadro 6 apresenta os Requisitos Funcionais (RF) do *framework* e o Quadro 7 apresenta os Requisitos Não Funcionais (RNF).

Quadro 6 - Requisitos funcionais do *framework*

Requisitos Funcionais do framework.
RF01: O *framework* deverá manter equipes
RF02: O *framework* deverá manter estatísticas das equipes.
RF03: O *framework* deverá manter jogadores.
RF04: O *framework* deverá manter estatísticas dos jogadores.
RF05: O *framework* deverá calcular a média geral das equipes a partir de uma estatística selecionada.
RF06: O *framework* deverá calcular a média das equipes comparando estatísticas historicamente.
RF07: O *framework* deverá calcular a média geral dos jogadores utilizando suas estatísticas avançadas.
RF08: O *framework* deverá calcular a média dos jogadores comparando estatísticas historicamente.

Fonte: Elaborado pelo autor.

Quadro 7 - Requisitos não funcionais do *framework*

Requisitos Não Funcionais do framework.
RNF01: O *framework* deverá ser desenvolvido utilizando a linguagem de programação C#.
RNF02: O *framework* deverá utilizar o ambiente Visual Studio para o desenvolvimento.
RNF03: O *framework* deverá suportar o upload de arquivos XLS e XLSX para atualização das estatísticas.
RNF04: O *framework* deverá utilizar o banco de dados não relacional MongoDB.

Fonte: Elaborado pelo autor.

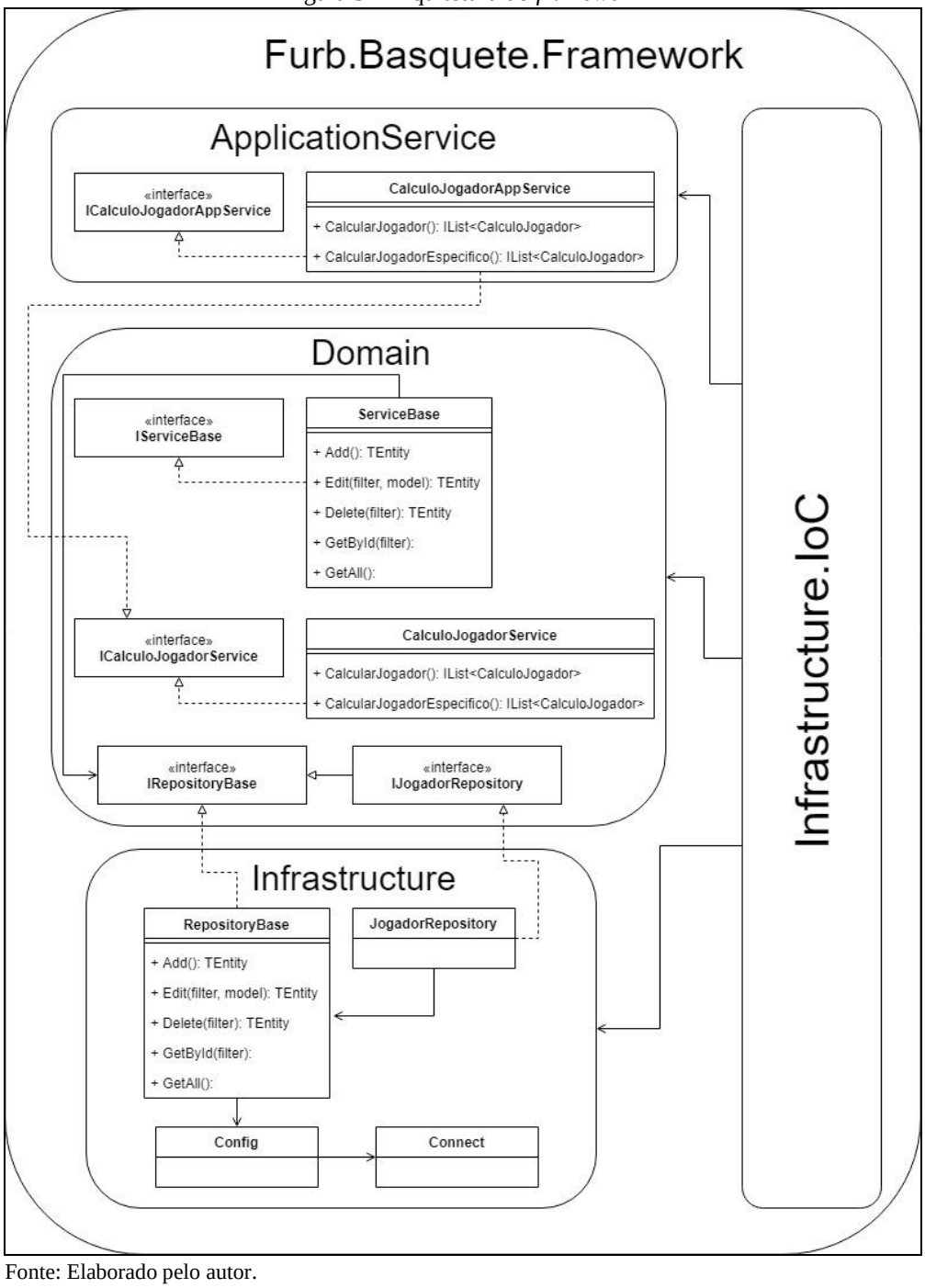
O *framework* foi desenvolvido utilizando o padrão de desenvolvimento conhecido como Injeção de Dependência (DI) e Inversão de Controle (IoC). Dessa forma, o desenvolvedor não instancia os objetos e sim implementa suas interfaces para chamar seus métodos. Utilizando essa abordagem de implementação o código fica mais livre sem precisar se preocupar com as dependências entre objetos. Por exemplo, uma classe X que possua um objeto Y em seu construtor, que por sua vez esse objeto Y possua um terceiro objeto Z como parâmetro no construtor. Quando for necessário instanciar um objeto da classe X, não precisa se preocupar com a dependência do objeto Z com o objeto Y, a arquitetura da injeção de dependência realiza esse trabalho. Lima (2017, p. 20) comenta que a utilização da injeção de dependência faz com que uma classe tenha uma dependência para uma interface cuja instância apropriada será recebida automaticamente. A DI serve para resolver automaticamente todas as dependências que uma classe possui em seu construtor sem a necessidade de informa-las toda vez que for instanciá-la (LIMA, 2017).

A Figura 3 mostra a arquitetura utilizada para o desenvolvimento do *framework*, sendo dividido em 4 partes e seguindo o padrão apresentado pelo DDD. A camada `ApplicationService` contém as interfaces e serviços concretos que irão realizar a comunicação do domínio com a aplicação que irá utilizar o *framework*. Quando a aplicação precisar chamar os métodos de cálculos dos jogadores basta implementar a interface `ICalculoJogadorAppService` que será implementada pela classe concreta `CalculoJogadorAppService`. A classe `CalculoJogadorAppService` contém uma propriedade injetada da interface `ICalculoJogadorService`, essa por sua vez é implementada na camada de domínio. A camada de domínio é responsável pelas regras de negócio, os cálculos e buscas, e possui uma interface base `IServiceBase` contendo os métodos mais genéricos para CRUD e persistência de informações que é implementada pela classe `ServiceBase`. A interface `ICalculoJogadorService` herda os métodos da classe `ServiceBase` e é

implementada pela classe `CalculoJogadorService`. Essa classe é responsável por buscar as informações no banco e realizar os cálculos matemáticos e comparações históricas. O *framework* disponibiliza para a aplicação a possibilidade de herdar os métodos genéricos de CRUD, porém a aplicação precisa seguir algumas regras. A primeira regra é criar uma classe com a herança da classe `Entity`, em seguida é necessário criar uma classe de serviços para essa classe herdando da classe `ServiceBase` e passando como parâmetro exigido a classe criada anteriormente.

A camada responsável por realizar a comunicação com o banco de dados é denominada de `Infrastructure`. Nela é realizada a implementação dos métodos de repositório de cada classe no banco e também as configurações do MongoDB. Existe a classe `RepositoryBase`, responsável por comunicar diretamente com o banco de dados. As demais classes de repositório herdam desta base informando a tabela correspondente no MongoDB. Em paralelo a estas camadas existe a `Infrastructure.IoC` que é a responsável por aplicar e configurar a injeção de dependência entre todas as camadas.

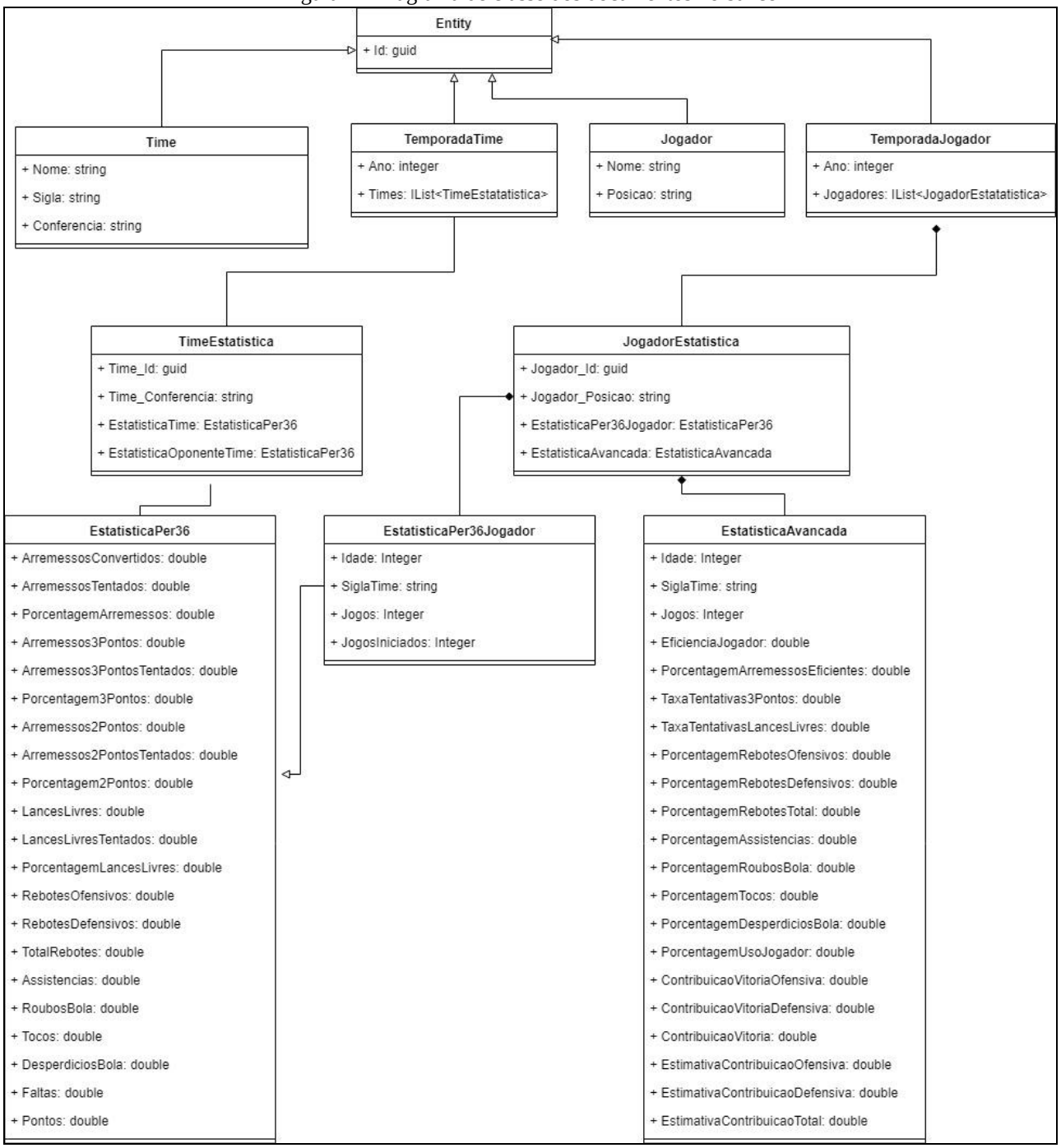
Figura 3 - Arquitetura do *framework*



Fonte: Elaborado pelo autor.

Na Figura 4 pode ser visualizado o diagrama de classe correspondente as entidades gravadas no banco de dados não relacional. Para o desenvolvimento das estatísticas de basquete foram utilizadas quatro classes que foram gravadas no MongoDB. As classes `Time`, `TemporadaTime`, `Jogador` e `TemporadaJogador` seguem o padrão descrito anteriormente herdando da classe `Entity`. As outras classes são usadas para possibilitar o armazenamento dos dados históricos. O dicionário de dados está descrito no APÊNDICE A .

Figura 4 - Diagrama de classe dos documentos no banco



Fonte: Elaborado pelo autor.

Os dados armazenados no banco de dados MongoDB são em formato JSON. O Quadro 8 pode-se visualizar um exemplo das informações salvas pelo *framework* referente a entidade `TemporadaJogador`. A classe `TemporadaJogador` possui objetos da classe `JogadorEstatistica`, que por sua vez possui objetos das classes `EstatisticaPer36Jogador` e `EstatisticaAvancada`. Com essas ligações é possível construir um objeto JSON conforme a figura ilustra.

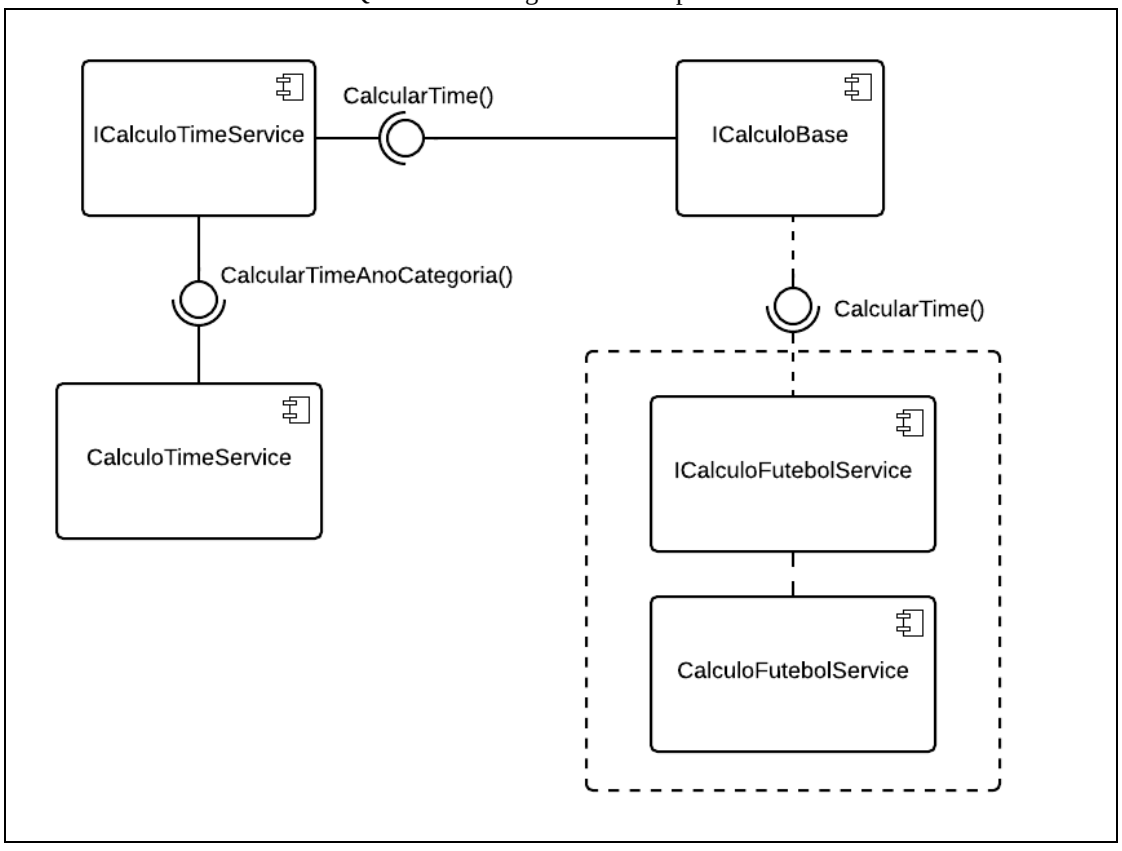
Quadro 8 - Registro gravado no MongoDB

```
"_id" : NUUID("a42b44b0-efe1-41a8-bc2e-550b30ddea0c"),
    "Ano" : 2018,
    "Jogadores" : [
        {
            "Jogador_ID" : NUUID("0b56bae0-37e8-4ec7-a19d-ce318cf51035"),
            "Jogador_Posicao" : "PG",
            "EstatsticaPer36" : {
                "ArremessosConvertidos" : 5.3,
                "ArremessosTentados" : 13.1,
                "PorcentagemArremessos" : 40.6,
```

Fonte: Elaborado pelo autor.

O *framework* fornece uma estrutura para os cálculos baseados na interface `ICalculoBase`, que fornece um método genérico `CalcularTime`. O *framework* disponibiliza a interface `ICalculoTimeService`, a qual herda da interface base `ICalculoBase` e disponibiliza o método `CalcularTimeAnoCategoria`. O *framework* também disponibiliza a classe concreta `CalculoTimeService` como implementação da interface de cálculo do time. Para uma aplicação utilizar a interface base de cálculo será necessário criar uma interface herdando de `ICalculoBase` e uma classe concreta implementando esta interface. Por exemplo, uma aplicação voltada para o futebol poderá criar a interface `ICalculoFutebolService` que irá herdar da interface `ICalculoBase`, consequentemente a interface conterá o método `CalcularTime`. O desenvolvimento será realizado na classe concreta `CalculoFutebolService` implementando a interface `ICalculoFutebolService`. O Quadro 9 apresenta estas informações no formato de diagrama de componentes.

Quadro 10 - Diagrama de componentes



Fonte: Elaborado pelo autor.

O *framework* foi desenvolvido pensando na escalabilidade do software, desta forma foi necessária a construção de uma interface base para fornecer os métodos básicos para as outras classes concretas utilizarem. A interface fornece os mesmos métodos para adicionar, editar, encontrar, buscar, listar e excluir de maneira síncrona e assíncrona. O desenvolvedor pode escolher entre qual for o mais indicado para cada situação. Os métodos síncronos são os mais comuns e utilizados no desenvolvimento, enquanto a opção assíncrona será recomendada para alguma operação que demanda muito tempo para ser executada. Como o *framework* utiliza cálculos de temporadas analisando estatísticas em

alguns casos faz-se necessário o uso de métodos assíncronos. No Quadro 11 pode-se ver a implementação da interface com os métodos de maneira síncrona e assíncrona.

Quadro 11 - Interface Base do *framework*

```
1  public interface IServiceBase<TEntity> : IDisposable where TEntity : class, IEntity
2  {
3      TEntity Add(TEntity model);
4      IEnumerable<TEntity> Add(IEnumerable<TEntity> models);
5      Task<TEntity> AddAsync(TEntity model);
6      Task<IEnumerable<TEntity>> AddAsync(IEnumerable<TEntity> models);
7
8      bool Edit(Expression<Func<TEntity, bool>> filter, TEntity model);
9      Task<bool> EditAsync(Expression<Func<TEntity, bool>> filter, TEntity model);
10
11     bool Update(Expression<Func<TEntity, bool>> filter, UpdateDefinition<TEntity> update);
12     bool UpdateAll(Expression<Func<TEntity, bool>> filter, UpdateDefinition<TEntity> update);
13     Task<bool> UpdateAsync(Expression<Func<TEntity, bool>> filter, UpdateDefinition<TEntity> update);
14     Task<bool> UpdateAllAsync(Expression<Func<TEntity, bool>> filter, UpdateDefinition<TEntity> update);
15
16     TEntity Find(Expression<Func<TEntity, bool>> filter);
17     Task<TEntity> FindAsync(Expression<Func<TEntity, bool>> filter);
18
19     IEnumerable<TEntity> GetAll();
20     IEnumerable<TEntity> GetAll(Expression<Func<TEntity, bool>> filter);
21     IEnumerable<TEntity> GetAll<Tkey>(Expression<Func<TEntity, bool>> filter, Expression<Func<TEntity, Tkey>> orderBy);
22     Task<IList<TEntity>> GetAllAsync();
23     Task<IList<TEntity>> GetAllAsync(Expression<Func<TEntity, bool>> filter);
24     Task<IList<TEntity>> GetAllAsync<Tkey>(Expression<Func<TEntity, bool>> filter, Expression<Func<TEntity, Tkey>> orderBy);
25
26     IList<TEntity> List<Tkey>(Expression<Func<TEntity, Tkey>> orderBy, Expression<Func<TEntity, bool>> filter = null);
27     Task<IList<TEntity>> ListAsync<Tkey>(Expression<Func<TEntity, Tkey>> orderBy, Expression<Func<TEntity, bool>> filter = null);
28
29     bool Delete(Expression<Func<TEntity, bool>> filter);
30     Task<bool> DeleteAsync(Expression<Func<TEntity, bool>> filter);
31 }
```

Fonte: Elaborado pelo autor.

A interface `IServiceBase` exige um parâmetro `TEntity`. Esse parâmetro obrigatoriamente precisa ser uma classe e herdar da interface `IEntity`, caso contrário não é possível implementar os métodos da interface `IServiceBase`. Como implementação da interface base, o *framework* possui a classe concreta `ServiceBase`. A classe concreta implementa todas as funções necessárias para persistência no banco de dados, ou seja, toda classe herdada de `ServiceBase` terá implementações padrões para os métodos fornecidos pela interface `IServiceBase`. A classe concreta exige o mesmo tipo de parâmetro `TEntity` da interface base. No Quadro 12 pode-se ver um exemplo de implementação da classe e também a utilização da injeção de dependência no construtor da classe, instanciando um objeto da interface `IRepositoryBase`.

Quadro 12 – Classe ServiceBase

```
1  public class ServiceBase<TEntity> : IServiceBase<TEntity> where TEntity : class, IEntity
2  {
3      private readonly IRepositoryBase<TEntity> _repository;
4
5      public ServiceBase(IRepositoryBase<TEntity> repository)
6      {
7          _repository = repository;
8      }
9
10     public TEntity Add(TEntity model)
11     {
12         return _repository.Add(model);
13     }
14
15     public IEnumerable<TEntity> Add(IEnumerable<TEntity> models)
16     {
17         return _repository.Add(models);
18     }
```

Fonte: Elaborado pelo autor.

O *framework* disponibiliza a interface `ICalculoBaseService`. Esta exige duas classes como parâmetros, formando assim o contrato de utilização. A primeira é a classe de `Command`, responsável pela entrada das informações e a segunda é classe de `Response`, a saída dos dados. Essa interface disponibiliza o método `CalcularTime`, podendo ser implementado da maneira mais apropriada para a classe concreta. O Quadro 13 apresenta a interface `ICalculoBaseService`, entre as linhas 1 e 6, e também a interface `ICalculoTimeService`, entre as linhas 9 e 13. A interface `ICalculoTimeService` herda de `ICalculoBaseService`, porém para utilizá-la foi necessário passar os argumentos `CalculoTimeCommand` e `CalculoTimeResponse`.

Quadro 13 – Interfaces de cálculo

```
public interface ICalculoBaseService<TCommand, TResponse>
    where TCommand : CommandBase
    where TResponse : ResponseBase
{
    IList<TResponse> CalcularTime(TCommand calculoTime);
}


public interface ICalculoTimeService : ICalculoBaseService<CalculoTimeCommand, CalculoTimeResponse>
{
    IList<CalculoTimeResponse> CalcularTime(CalculoTimeCommand calculoTime);
    IList<CalculoTimeAnoCategoria> CalcularTimeAnoCategoria(int anoInicio, int anoFim, Time time, TipoCategoria categoria);
}
```

Fonte: Elaborado pelo autor.

A inserção de dados no banco de dados foi implementada realizando a leitura de arquivos `XLS` ou `XLSX` por parte da aplicação. No entanto a aplicação fica livre para fazer a leitura com qualquer tipo de arquivo. O *framework* recebe como parâmetros de entrada uma lista das classes `Commands` exigidas para a importação das estatísticas. A construção desta lista fica ao critério da aplicação. O Quadro 14 mostra a interface `ITemporadaJogadorAppService` com o método responsável por adicionar as estatísticas dos jogadores. Como parâmetro do método é necessária uma lista de `TemporadaJogadorCommand`, que pode ser preenchida como a aplicação desejar.

Quadro 14 - Inserção de dados

```
public interface ITemporadaJogadorAppService : IAppServiceBase<TemporadaJogador>
{
    void AdicionarTemporadaJogador(IList<TemporadaJogadorCommand> jogadores);
    TemporadaJogadorResponse ObterEstatisticaJogador(Guid idJogador);
}
```

Fonte: Elaborado pelo autor.

O *framework* já possui a implementação de classes concretas de `Jogador` e `Time` para as estatísticas de basquete, ambas as classes seguem o padrão adotado pelo *framework* de herança das classes exigidas e implementam as interfaces base incluindo outros métodos específicos para cada classe. A realização dos cálculos e buscas das estatísticas foram baseados no Quadro 15. A tabela A representa as estatísticas mais convencionais tanto para Time quanto para Jogador, enquanto a tabela B representa as estatísticas avançadas, estando disponível apenas para os jogadores. A fórmula para chegar nos resultados de cada registro da tabela não são de responsabilidade do *framework*.

Quadro 15 - Estatísticas do basquete

Estatística	Sigla
Arremessos Convertidos	AC
Arremessos Tentados	AT
Porcentagem Arremessos	%AC
Arremessos 3 Pontos	3A
Arremessos 3 Pontos Tentados	3AT
Porcentagem 3 Pontos	3A%
Arremessos 2 Pontos	2ª
Arremessos 2 Pontos Tentados	2AT
Porcentagem 2 Pontos	2A%
Lances Livres	LL
Lances Livres Tentados	LLT
Porcentagem Lances Livres	LL%
Rebotes Ofensivos	RO
Rebotes Defensivos	RD
Total Rebotes	TR
Assistências	A
Roubos Bola	RB
Tocos	TCS
Desperdícios Bola	DB
Faltas	FTS
Pontos	PTS

(A)

Estatística Avançada	Sigla
Eficiência Jogador	EJ
Porcentagem Arremessos Eficientes	AE%
Taxa Tentativas 3 Pontos	TT3
Taxa Tentativas Lances Livres	TTLL
Porcentagem Rebotes Ofensivos	RO%
Porcentagem Rebotes Defensivos	RD%
Porcentagem Rebotes Total	TR%
Porcentagem Assistências	A%
Porcentagem Roubos Bola	RB%
Porcentagem Tocos	TCS%
Porcentagem Desperdícios Bola	DB%
Porcentagem Uso Jogador	US%
Contribuição Vitoria Ofensiva	CVO
Contribuição Vitoria Defensiva	DVD
Contribuição Vitoria	CV
Estimativa Contribuição Ofensiva	ECO
Estimativa Contribuição Defensiva	ECD
Estimativa Contribuição Total	ECT

(B)

Fonte: Elaborado pelo autor.

O *framework* realizou os cálculos de acordo com as estatísticas para cada tipo de classe. As estatísticas dos times foram o período das temporadas, a escolha da conferência, o critério, ou seja, os cálculos baseados nas estatísticas do time ou as estatísticas dos outros times quando enfrentaram aquele time, e pôr fim a categoria, uma das linhas da tabela A. Para as estatísticas dos jogadores foram utilizadas variáveis semelhantes, com a adição da escolha por posição e estatística avançada. O critério de posição é importante para que as estatísticas sejam mais próximas, pois não é uma

comparação justa aplicar os cálculos entre jogadores de posições e tamanhos distintos. A partir dos dados obtidos, o *framework* realiza a busca no banco de dados e faz os cálculos da média entre times ou jogadores. A média calculada pode ser histórica, comparando times e jogadores da atualidade com o passado, como também o cálculo de jogador específico, comparando as estatísticas avançadas com jogadores de mesma posição.

4 RESULTADOS

O desenvolvimento deste trabalho possibilitou a criação de um *framework* para análise estatísticas do basquete comparando os jogadores e times historicamente e individualmente, avaliando a média da liga e por cada posição e apresentando os resultados. Comparando o *framework* desenvolvido com os trabalhos correlatos apresentados na subseção 2.5 é possível observar algumas semelhanças e diferenças. O *framework* desenvolvido é o único que apresenta a arquitetura em camadas para que seja possível disponibilizar as funcionalidades para outras aplicações. Por outro lado, o projeto de Cao (2012) é o único com a possibilidade de prever resultados de basquete. O trabalho de Ivankovic et al. (2010) não utiliza as estatísticas avançadas do basquete para analisar e comparar os resultados, uma característica que os outros possuem, deixando as informações mais completas. Todos os trabalhos apresentados utilizam-se da comparação históricas para chegar no resultado desejado, uma maneira de analisar os dados atuais é comparar os mesmos com outras temporadas para obter melhores conclusões. O Quadro 16 apresenta estas informações de forma resumida.

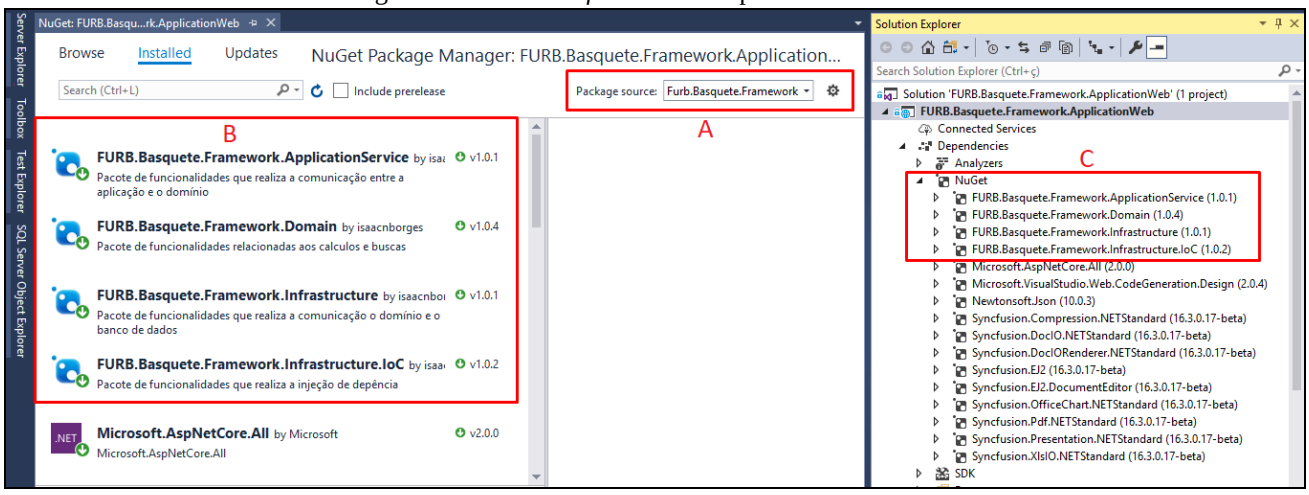
Quadro 16 - Comparação trabalhos correlatos

Características	**Este**	**Ivankovic et al. (2010)**	**FiveThirtyEight (2018)**	**Cao (2012)**
Utiliza estatísticas avançadas	Sim	Não	Sim	Sim
Comparação histórica	Sim	Sim	Sim	Sim
Prever resultados	Não	Não	Não	Sim
Disponibiliza funcionalidades para outras aplicações	Sim	Não	Não	Não
Arquiteturas em camadas	Sim	Não	Não	Não

Fonte: Elaborado pelo autor.

O *framework* fornece como proposta um padrão de desenvolvimento para que as aplicações o utilizem e diminua as classes e serviços para consultar as estatísticas, consequentemente organizando a aplicação. Para utilizar as funcionalidades é preciso criar uma aplicação em C# e adicionar os projetos referentes ao *framework* no pacote NuGet da solução. Os pacotes do *framework* foram disponibilizados em uma pasta localmente para fins de testes, mas é possível distribuir em um servidor dedicado ou mesmo na base de dados oficial do NuGet. A Figura 5 apresenta a aplicação web desenvolvida no Visual Studio utilizando os pacotes oferecidos pelo *framework*. O destaque A mostra o `package source`, em que é possível escolher quais diretórios o NuGet irá pesquisar os pacotes para adicionar na aplicação. Neste exemplo é selecionado a pasta `Furb.Basquete.Framework`. Com a pasta selecionada é possível visualizar os projetos do *framework* desenvolvidos e disponibilizados para uso, sendo possível visualizá-los no destaque B. Por fim, na parte C são apresentadas todas as referências NuGet do projeto da aplicação, como os arquivos da Microsoft, os componentes *Syncfusion* e os projetos do *framework*.

Figura 5 - Pacotes do *framework* disponíveis no NuGet



Fonte: Elaborado pelo autor.

4.1 APLICAÇÃO DO FRAMEWORK

Com o objetivo de testar e analisar a utilização do *framework* foi construída uma aplicação ASP.NET Core com as referências do *framework*. A aplicação fica disponível para o usuário em ambiente web e seu desenvolvimento segue

o padrão Model View Controller (MVC) adotado pelo Visual Studio. A configuração do banco de dados é responsabilidade da aplicação. O arquivo `appsettigs.json` é adicionado a *tag* `MongoDB` informando qual a banco de dados e a `string` de conexão (`ConnectionStrings`). A Figura 6 demonstra a utilização da *tag* `MongoDB`, utilizada na `linha 8`. A `linha 9` apresenta a *tag* `Database` com o nome do banco de dados, nesse caso FURB-Basquete. A `linha 10` é configurado a `ConnectionStrings`, endereço do servidor em que se encontra o banco de dados. Cada aplicação fica responsável por informar o banco de dados e a `connectionStrings`, enquanto o *framework* realiza a comunicação e persistência das informações.

Figura 6 - Configuração banco de dados

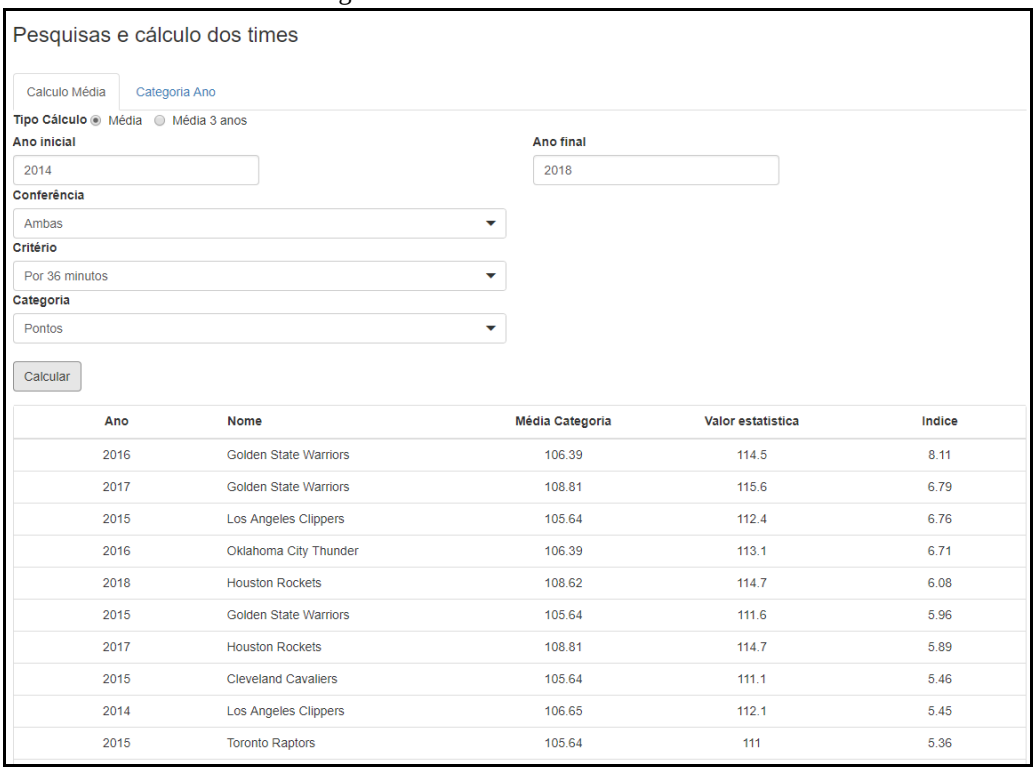
```
1  {
2    "Logging": {
3      "IncludeScopes": false,
4      "LogLevel": {
5        "Default": "Warning"
6      }
7    },
8    "MongoDB": {
9      "Database": "FURB-Basquete",
10     "ConnectionStrings": "mongodb://localhost:27017"
11   },
12   "AllowedHosts": "*"
13 }
```

Fonte: Elaborado pelo autor.

A aplicação foi desenvolvida com o foco nas estatísticas de jogadores e times de basquete, uma vez que o *framework* possui as classes concretas que realizam a implementação das mesmas, com isso foi possível construir uma interface gráfica utilizando tabelas e gráficos para melhor visualização das informações para o usuário. Os dados importados para as estatísticas dos jogadores e times foram das temporadas de 2000 até 2018 e foram obtidos do site Basketball-Reference.

A Figura 7 apresenta o cálculo histórico dos times, o usuário escolher entre a média anual da temporada ou média a cada 3 anos, apresentado no tipo de cálculo, e deve informar o período da pesquisa. Além disso é possível escolher a conferência das equipes, o critério pode variar entre as estatísticas da equipe ou estatísticas dos oponentes contra o time em questão, por último é escolhido qual a categoria deseja aplicar o filtro.

Figura 7 - Cálculo histórico dos times



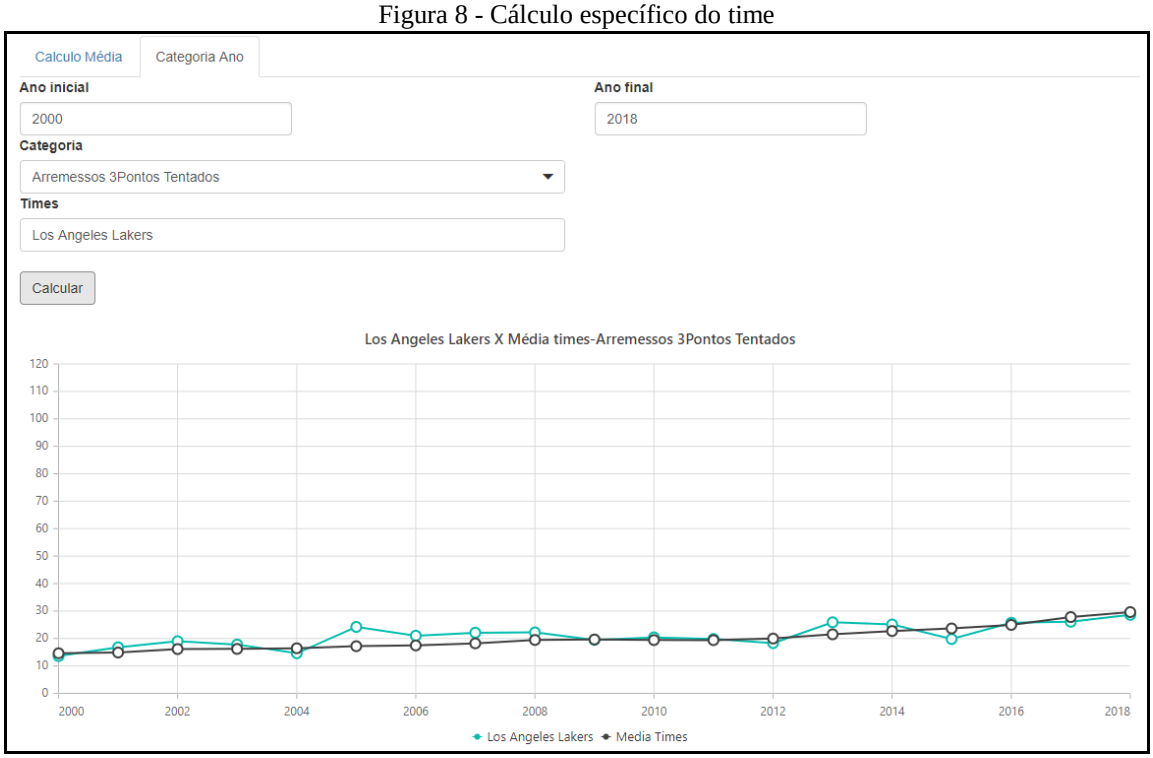
Ano	Nome	Média Categoria	Valor estatistica	Indice
2016	Golden State Warriors	106.39	114.5	8.11
2017	Golden State Warriors	108.81	115.6	6.79
2015	Los Angeles Clippers	105.64	112.4	6.76
2016	Oklahoma City Thunder	106.39	113.1	6.71
2018	Houston Rockets	108.62	114.7	6.08
2015	Golden State Warriors	105.64	111.6	5.96
2017	Houston Rockets	108.81	114.7	5.89
2015	Cleveland Cavaliers	105.64	111.1	5.46
2014	Los Angeles Clippers	106.65	112.1	5.45
2015	Toronto Raptors	105.64	111	5.36

Fonte: Elaborado pelo autor.

A tabela mostra a classificação das melhores equipes ordenas pelo índice, estatística calculada a partir da média da categoria anual com o valor da estatística em si. Como é possível observar, a equipe do Golden State Warriors do ano de 2017 possuiu um ataque de 115,6 pontos por jogo, um valor maior que os 114,5 pontos do mesmo Golden State Warriors da temporada de 2016. Isso faz que com a equipe de 2016 tenha um ataque superior a equipe de 2017 em dados brutos, entretanto o *framework* organiza de diferente forma. A média de pontos por jogo da temporada de 2016 de todas as equipes foi de 106,39 pontos, enquanto em 2017 foi de 108,81, isso faz com que o time dos Warriors possua um índice de 8,11 em relação a média de pontos da temporada 2016, superando os 6,79 que a equipe conseguiu em 2017. O índice é calculado a partir da subtração entre a média da categoria e o valor da estatística. Com isso é possível entender que o ataque de todas as equipes melhorou no ano de 2017, fazendo com que o ataque de 2016 seja mais impressionante no contexto em que se encontra.

A Figura 8 mostra a busca histórica entre os anos de 2000 até 2018 do time Los Angeles Lakers, escolhendo a categoria de arremessos de 3 pontos tentados. Essa comparação é importante destacar o aumento das bolas 3 pontos ao longo da última década por parte de toda a liga, saindo de 14,6 arremessos no início do século para 29,6 na temporada de 2018. Isso mostra para qual lado a liga está caminhando, um aumento considerável nos arremessos de 3 pontos, gerando com isso mais pontos por jogo.

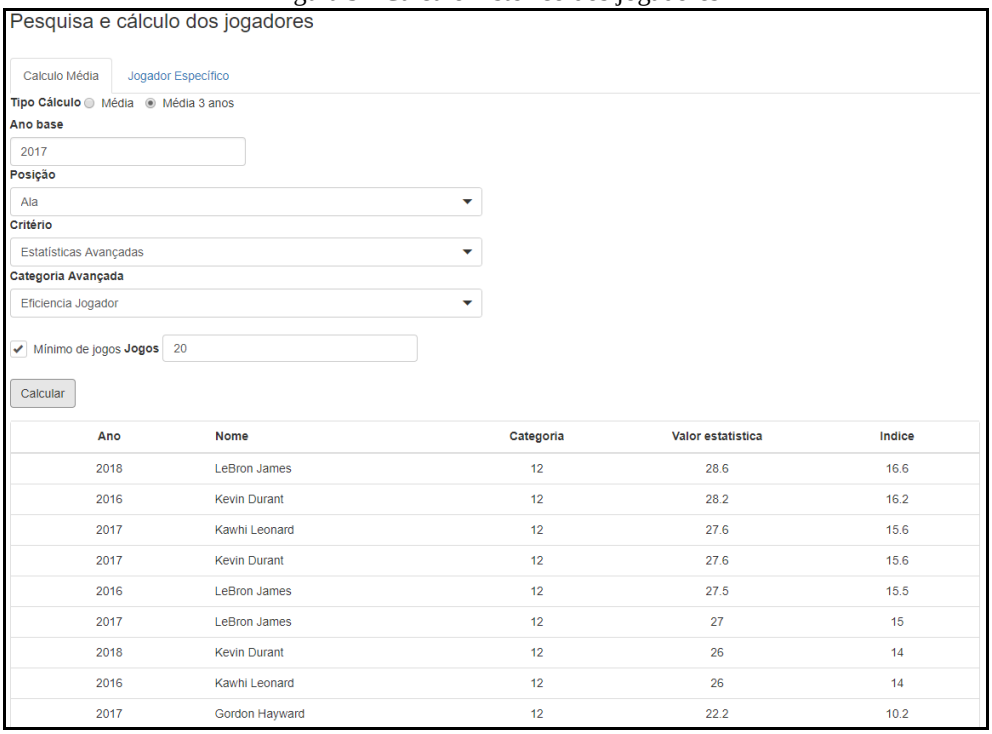
Figura 8 - Cálculo específico do time



Fonte: Elaborado pelo autor.

A Figura 9 mostra as estatísticas históricas para os jogadores. A tela mostra muita semelhança com o cálculo histórico dos times, porém neste exemplo é utilizado o tipo de cálculo “média 3 anos” para as buscas. A pesquisa usa como base a temporada de 2017, isso faz com que o *framework* realize a pesquise dos anos de 2016 até 2018 e faça a média entre essas três temporadas. Para os jogadores é utilizada a posição como parâmetro determinante nas buscas, uma vez que a posição no basquete tende a diferenciar os jogadores por algumas características. O exemplo utiliza a posição de Ala e calcula a estatística avançada de eficiência do jogador. Essa estatística é um cálculo matemático considerando várias realizações do jogador em quadra, tanto positivas (pontos, assistências, rebotes) como negativas (faltas, desperdícios de bola). O *framework* não realiza esse cálculo, apenas obtêm a informação importada anteriormente, uma vez que a fórmula já existe.

Figura 9 - Cálculo histórico dos jogadores

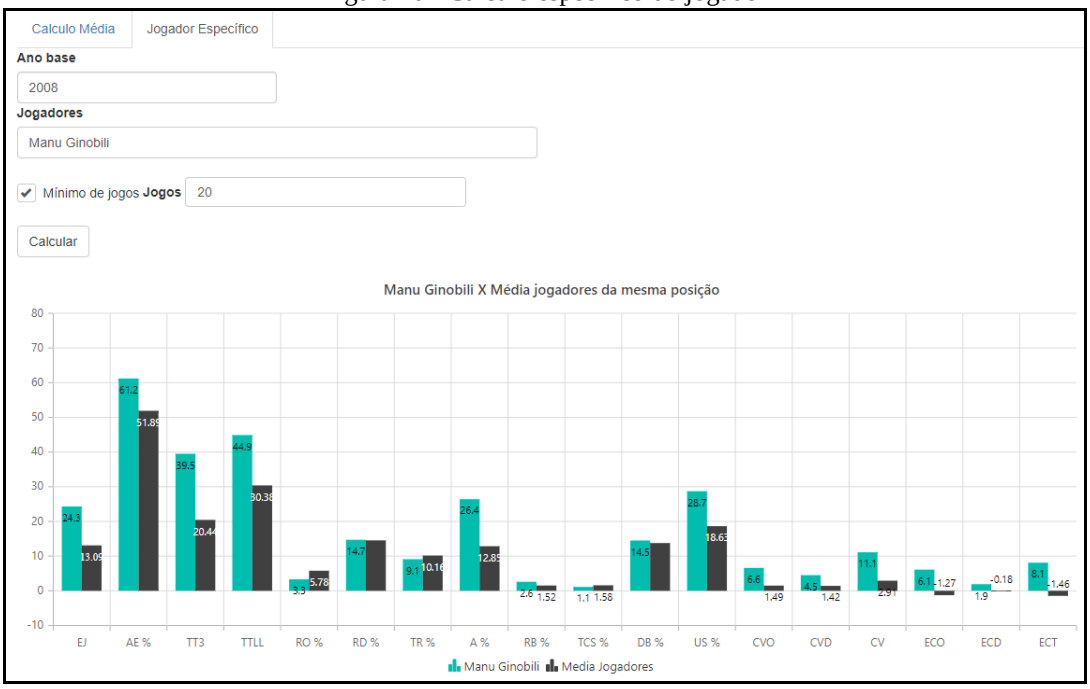


Ano	Nome	Categoria	Valor estatistica	Indice
2018	LeBron James	12	28.6	16.6
2016	Kevin Durant	12	28.2	16.2
2017	Kawhi Leonard	12	27.6	15.6
2017	Kevin Durant	12	27.6	15.6
2016	LeBron James	12	27.5	15.5
2017	LeBron James	12	27	15
2018	Kevin Durant	12	26	14
2016	Kawhi Leonard	12	26	14
2017	Gordon Hayward	12	22.2	10.2

Fonte: Elaborado pelo autor.

Para demonstrar as estatísticas avançadas dos jogadores de forma detalhada existe o cálculo específico de cada jogador. A Figura 10 apresenta o cálculo de todas as estatísticas avançadas de um jogador específico com a média da liga no ano desejado. Com essas informações é possível visualizar o rendimento do jogador e endender em quais circustancias o jogador destaca-se positavemente ou negativamente e se faz sentido mantê-lo na equipe ou não.

Figura 10 - Cálculo específico do jogador

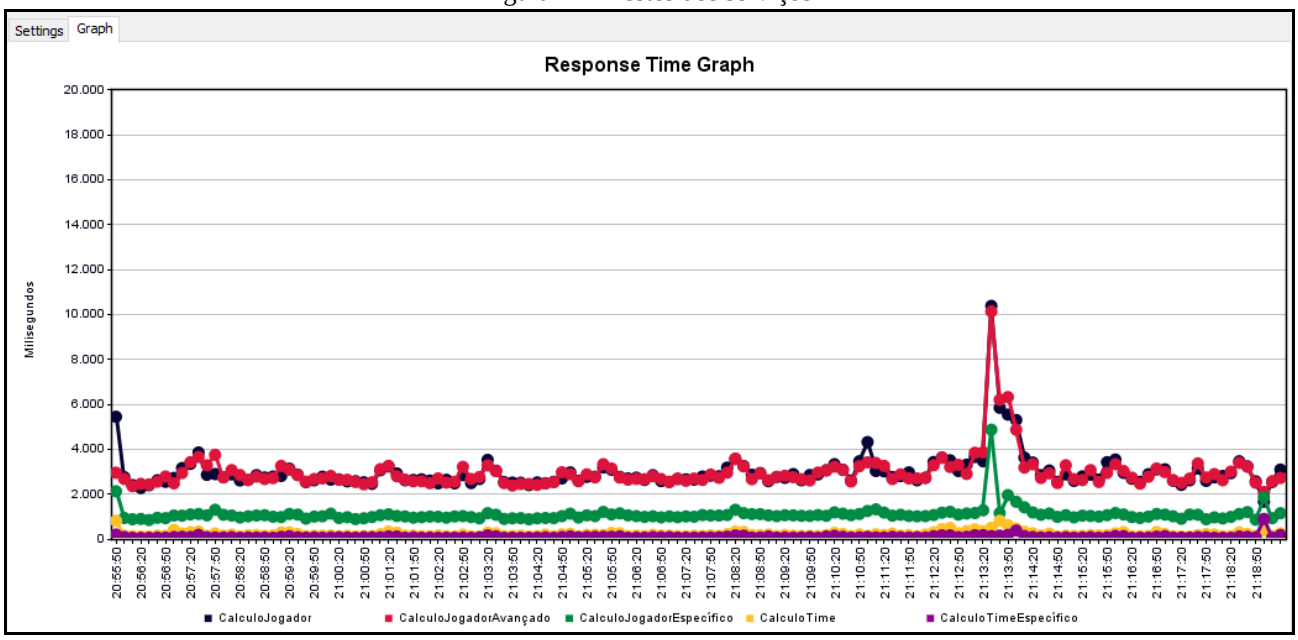


Fonte: Elaborado pelo autor.

Uma das maneiras de testar a performance do *framework* foi utilizando a ferramenta JMeter, realizando uma bateria de testes de carga para avaliar a quantidade de tempo em que os dados eram retornados. Todos os testes foram direcionados para a aplicação, no qual é possível acessar através de web API. O cenário proposto simulou 4 usuários realizando 200 requisições em um intervalo de 5 segundos cada, resultando em um total de 5000 requisições. Importante ressaltar que os testes foram realizados em um ambiente local, ou seja, com todos os serviços e banco de

dados rodando na mesma máquina, e os testes foram com dados pré-determinados. O resultado pode ser visualizado na Figura 11 em que é apresentado os cinco cenários dos cálculos com o tempo medido em milissegundos.

Figura 11 - Testes dos serviços



Fonte: Elaborado pelo autor.

Os testes iniciaram as 20 horas 55 minutos e finalizaram as 21 horas e 18 minutos. Como esperado, os cálculos de jogadores levaram mais tempo comparado aos de times, isto porque existem apenas 30 times enquanto a quantidade de jogadores gira em torno de 450 por temporada. Os casos mais demorados foram concluídos com um pouco mais de 10.000 milissegundos, algo em torno de 10 segundos para a aplicação obter os dados, o *framework* receber esses dados, buscar no banco de dados não relacional, realizar os cálculos e formatar a resposta para a aplicação.

5 CONCLUSÕES

O principal objetivo deste trabalho foi o desenvolvimento de um *framework* que possa auxiliar no desenvolvimento de aplicações para análise estatísticas voltadas ao basquete. Ao fazer uso das estatísticas avançadas foi possível trabalhar com comparações históricas e cálculos específicos para analisar e montar os resultados. O *framework* facilita o desenvolvimento em várias ocasiões fazendo com que a aplicação herde ou implemente as funções já construídas pelo *framework*.

Os objetivos específicos referentes a disponibilização de métodos concretos para identificar as características dos jogadores e equipes foram atendidos. O usuário pode informar os critérios que mais se adequam a pesquisa e o *framework* realiza as buscas. Os objetivos relacionados aos cálculos e buscas históricas de jogadores e times específicos foram atendidos, isso pode-se afirmar, pois os cálculos foram baseados nas estatísticas normais e avançadas do basquete. Os times são comparados historicamente com a média geral da liga na categoria determinada, enquanto os jogadores são comparados com a média da liga dos demais jogadores da mesma posição em todas as estatísticas avançadas. O objetivo específico referente ao desenvolvimento de uma aplicação para utilizar o *framework* também foi atendido, pois a aplicação foi desenvolvida utilizando o Visual Studio e usando a tecnologia ASP.NET Core.

A ferramenta Visual Studio com a linguagem C# mostrou-se eficaz no desenvolvimento por conter as funcionalidades para a utilização da injeção de dependência e o padrão arquitetural DDD. Tais técnicas cumpriram o papel no âmbito de conceituação, assim como na parte de implementação. O DDD está presente na arquitetura do *framework* e foi importante para isolar cada parte do desenvolvimento, enquanto a injeção de dependência está em praticamente todas as classes e interfaces do projeto, com o intuído de isolar as dependências de cada classe e objeto. Para a persistência dos dados optou-se pelo uso do banco de dados MongoDB, um banco de dados não relacional que se mostrou eficiente para buscar as informações com velocidade, uma vez que não existem ligações entre as tabelas.

Como vantagens, o *framework* proporciona a abstração no desenvolvimento de classes e interfaces para análise de estatísticas do basquete por parte da aplicação, sendo que todo o código relacionado a isso fica isolado nos projetos internos do *framework*. A aplicação não precisa se preocupar em como os dados serão gravados ou buscados no banco de dados, uma vez que esta parte do código também está abstraída no *framework*. Como desvantagens é possível afirmar que não foi aplicada uma generalização maior no desenvolvimento, ou seja, várias partes do *framework* poderiam ser mais genéricas e abstratas, facilitando assim o desenvolvimento por parte das aplicações. Outro ponto

negativo está no fato do *framework* não disponibilizar vários tipos de cálculos e buscas, sendo focado apenas na utilização da média entre jogadores e times.

Contudo, apesar das limitações, todos os objetivos foram concluídos, pois é possível gerar um padrão de desenvolvimento, além de que várias das funções que seriam necessárias desenvolver foram acopladas dentro dos projetos do *framework*. Além disto, o desenvolvimento deste trabalho proporcionou ao acadêmico a aplicação de várias áreas aprendidas ao longo do curso de graduação, além de aprofundar os conhecimentos no esporte, principalmente no basquete e na utilização das estatísticas.

5.1 EXTENSÕES

Como sugestão de extensões deste trabalho sugere-se:

a) incluir outros esportes no desenvolvimento e aplicação das estatísticas no *framework*;
b) melhorar a abstração das classes no *framework*, fazendo com que seja possível generalizar mais as interfaces e classes e diminuir o código-fonte tanto do *framework* como das aplicações;
c) adicionar novos tipos de cálculos nas estatísticas, como o uso da mediana ou moda;
d) incluir novas variáveis para os cálculos de basquete, levando em conta outros contextos como salário e idade dos jogadores, e posição da equipe na tabela;
e) melhorar o desenvolvimento da interface com usuário, utilizar *frameworks* mais recentes como *angular* ou *react native*, e disponibilizar outas opções de gráficos para facilitar o entendimento do usuário.

REREFÊNCIAS

BASKETBALL, Usa. **Defining the Positions**. 2015. Disponível em: <http://www.usab.com/youth/news/2012/08/defining-the-positions.aspx>. Acesso em: 07 Nov. 2018.

CALEIRO, João Pedro. **Qual é o tamanho do esporte na economia?**. 2014. Disponível em: <http://exame.abril.com.br/economia/qual-e-o-tamanho-da-importancia-do-esporte-na-economia/>. Acesso em: 19 Out. 2018.

CANAN, Felipe; MENDES, José Carlos; da SILVA, Rogério Vaz. **Análise estatística no basquetebol de base**: perfil do Campeonato Paranense de Basquetebol masculino Sub-17. 2015. 14p. Trabalho de conclusão de curso (Bacharelado em Educação Física) Universidade Estadual do Oeste do Paraná, Paraná.

CAO, Chenjie. **Sports Data Mining Technology Used in Basketball Outcome Prediction**. 2012. 106 f. Dissertação (Mestrado) - Curso de Data Analytics, Computing Center, Dublin Institute Of Technology, Dublin, 2012.

COBOS, Paulo. **Fora das quadras, como donos ou consultores, geeks ditam rumos na NBA**. 2010. Disponível em: <https://esporte.uol.com.br/basquete/ultimas-noticias/2010/04/04/fora-das-quadras-como-donos-ou-consultores-geeks-ditam-rumos-na-nba.jhtm>. Acesso em: 17 Nov. 2018.

FAVERO, Luiz Paulo; BELFIORE Patricia. **Manual de Análise de dados**: Estatística e modelagem multivariada com Excel, SPSS e Stata. Rio de Janeiro. Elsevier Brasil. 2017.

FAYAD, M.E; SCHMIDT, D.C.; JOHNSON, R.E. **Building Application Frameworks**. New York: Willey, 1999.

FIVETHIRTYEIGHT, **CARMELO NBA Player Projections**. 2017. Disponível em: <https://projects.fivethirtyeight.com/carmelo/>. Acesso em: 18 Dez 2018.

FRANCO, Jaqueline Rissá; MATTOS, Karla Marturelli; DOLL, Luciano Mathias; ALMEIDA, João. **Domain-Driven Design e Test-Driven Development**. 2010. Disponível em: < http://www.5eetcg.uepg.br/Anais/artigospdf/50021_vf1.pdf>. Acesso em: 03 Nov. 2018.

FERREIRA, Juliano. **8 tendências do Big Data em 2017**. 2017. Disponível em: <http://www.bigdatabusiness.com.br/tendencias-big-data-2017/>. Acesso em: 19 Out. 2018.

GREGORI, Reinaldo. **O que é análise estatística?** 2015. Disponível em: <https://www.linkedin.com/pulse/o-que-%C3%A9-an%C3%A1lise-estat%C3%ADstica-reinaldo-gregori>. Acesso em: 31 Out. 2018.

HEKIMA. **Saiba como a NBA usa Big Data para otimizar a experiência do cliente**. 2016. Disponível em: <http://www.bigdatabusiness.com.br/saiba-como-a-nba-usa-big-data-para-otimizar-a-experiencia-do-cliente/>. Acesso em: 19 Nov. 2018.

_____. **Como soluções de Big Data podem ajudar atletas e esportistas**. 2015 Disponível em: <http://www.bigdatabusiness.com.br/como-solucoes-de-big-data-podem-ajudar-atletas-e-esportistas/>. Acesso em: 19 Nov. 2018

IANNI, Vinicius. Big Data: **Algumas definições e, sim, serve também para o pequeno negócio!** 2016. Disponível em: <http://www.devmedia.com.br/big-data-algumas-definicoes-e-sim-serve-tambem-para-o-pequeno-negocio/27527>. Acesso em: 30 Out. 2018.

IVANKOVIC, Zdravko; IVKOVIC, M.; MARKOSKI, Branko; RACKOVIC, Miloš; RADOSAV, Dragica: **Analysis of basketball games using neural networks**. CINTI, 11., 2010, Budapest. Proceedings… Budapest: [s.n.]. 2010. p. 1-6.

KATCHBORIAN, Pedro. **Big data e basquete**: uma história de sucesso. 2016. Disponível em: <https://iq.intel.com.br/big-data-e-basquete-uma-historia-de-sucesso/>. Acesso em: 03 Out. 2018.

LIMA, Fabio Silva. **Injeção de Dependência**: A Redenção do Simple Injector. 2017. Disponível em: <https://www.fabiosilvalima.net/dependencia-de-injecao-simple-injector/>. Acesso em: 03 Out. 2018.

_____. **Programação no mundo real**: Design Patterns. 2017.

MAGNUS, Thiago. **Big Data nos esportes**: bons resultados estão nos detalhes. 2018. Disponível em: <https://transformacaodigital.com/big-data-nos-esportes/>. Acesso em: 03 Out. 2018.

MAXCY, Joel; DRAYER, Joris. **Sports Analytics**: Advancing Decision Making Through Technology and Data. Philadelphia, PA: Temple University, 2014. 28 p

ROSA, Daniele Toniolo Dias. **Fundamentos de Estatística**. 2012. Disponível em: <http://www.fotoacustica.fis.ufba.br/daniele/planejamento/Aula2.pdf>. Acesso em: 31 Out. 2018.

ROSE JÚNIOR, Dante de; TRICOLI, Valmor. **Basquetebol Uma Visão Integrada entre Ciência e Prática**. Barueri: Manole, 2005.

SILVA, Ricardo Pereira. **Suporte ao desenvolvimento e uso de frameworks e componentes**. 2000. 262 f. Tese (Doutorado em Ciência da Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre.

SILVA, Ricardo Pereira; FREIBERGER, Evandro Cesar. Metrics to Evaluate the Use of Object Oriented Frameworks. **The Computer Jornal**, Grã-Bretanha, v. 9, n. 3, p.2-4, Maio. 2009.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. Ed. São Paulo: Pearson, 2008.

TARIFA, Alexandre; **Big Data**: descubra o que é e como usar na sua empresa. 2014. Disponível em: <https://endeavor.org.br/big-data-descubra-o-que-e-e-como-usar-na-sua-empresa/>. Acesso em: 27 Out 2018.

VIEIRA, Silvia; FREITAS, Armando. **O que é Basquete**. Rio de Janeiro: Casa da Palavra, 2006. 100 p.

APÊNDICE A

Este Apêndice apresenta o dicionário de dados entre o Quadro 17 e

Quadro 24 contento a descrição dos documentos do banco de dados. Os tipos de dados utilizados neste apêndice são:

a) `guid`: número inteiro de 128 bits;

b) `integer`: armazena números inteiros de tamanho normal;

c) `string`: armazena cadeira de caracteres de vários tamanhos;

d) `double`: armazena número decimais com duas casas decimais após a virgula;

e) `list`: tipo necessário para criar listas com os objetos internos de cada documento;

f) `object`: representação abstrata para criar objetos internos nos documentos, no desenvolvimento foram utilizadas classes concretas;

Quadro 17 – Documento `Jogadores`

`Jogadores` – armazena jogadores.		
CAMPO	**DESCRIÇÃO**	**TIPO**
Id	Identificador único.	`Guid`
Nome	Nome do jogador.	`Integer`
Posição	Posição do jogador.	`string`

Fonte: Elaborado pelo autor.

Quadro 18 – Documento `Times`

`Times` – armazena times.		
CAMPO	**DESCRIÇÃO**	**TIPO**
Id	Identificador único.	`Guid`
Nome	Nome do time.	`Integer`
Sigla	Sigla do time.	`string`
Conferencia	Conferência do jogador.	`string`

Fonte: Elaborado pelo autor.

Quadro 19 – Documento `TemporadaJogador`

`TemporadaJogador` – armazena a temporada dos jogadores.		
CAMPO	**DESCRIÇÃO**	**TIPO**
Id	Identificador único.	`Guid`
Ano	Ano de cada temporada.	`Integer`
Jogadores	Lista contendo as estatísticas dos jogadores.	`List`

Fonte: Elaborado pelo autor.

Quadro 20 – Documento `TemporadaTime`

`TemporadaTime` – armazena a temporada dos times.		
CAMPO	**DESCRIÇÃO**	**TIPO**
Id	Identificador único.	`Guid`
Ano	Ano de cada temporada.	`Integer`
Times	Lista contendo as estatísticas dos times.	`List`

Fonte: Elaborado pelo autor.

Quadro 21 - Objeto `JogadorEstatistica`

`JogadorEstatistica` – objeto interno do documento `TemporadaJogador`.		
CAMPO	**DESCRIÇÃO**	**TIPO**
Jogador_Id	Identificador do jogador.	`Guid`

Jogador_Posicao	Posição do jogador.	`String`
EstatisticaPer36	Estatísticas dos jogadores calculadas por 36 minutos de jogo.	`object`
EstatisticaAvancada	Estatísticas avançadas dos jogadores.	`object`

Fonte: Elaborado pelo autor.

Quadro 22 - Objeto `TimeEstatistica`

`TimeEstatistica` – objeto interno do documento `TemporadaTime`.		
CAMPO	**DESCRIÇÃO**	**TIPO**
Time_Id	Identificador do time.	`Guid`
Time_Conferencia	Conferência do time.	`String`
EstatisticaPer36	Estatísticas dos times calculadas por 36 minutos de jogo.	`object`
EstatisticaOponenteTime	Estatísticas dos oponentes quando enfrentam determinado time.	`object`

Fonte: Elaborado pelo autor.

Quadro 23 - Objeto `EstatisticaPer36`

`EstatisticaPer36` – objeto interno contendo as estatísticas do documento `JogadorEstatistica` e `TimeEstatistica`.		
CAMPO	**DESCRIÇÃO**	**TIPO**
`ArremessosConvertidos`	Quantidade de arremessos totais convertidos por jogador e time.	`double`
`ArremessosTentados`	Quantidade de arremessos totais tentados por jogador e time.	`double`
`PorcentagemArremessos`	Percentual de arremessos totais convertidos por jogador e time.	`double`
`Arremessos3Pontos`	Quantidade de arremessos de 3 pontos convertidos por jogador e time.	`double`
`Arremessos3PontosTentados`	Quantidade de arremessos de 3 pontos tentados por jogador e time.	`double`
`Porcentagem3Pontos`	Percentual de arremessos de 3 pontos convertidos por jogador e time.	`double`
`Arremessos2Pontos`	Quantidade de arremessos de 2 pontos convertidos por jogador e time.	`double`
`Arremessos2PontosTentados`	Quantidade de arremessos de 2 pontos tentados por jogador e time.	`double`
`Porcentagem2Pontos`	Percentual de arremessos de 2 pontos convertidos por jogador e time.	`double`
`LancesLivres`	Quantidade de arremessos de lances livre convertidos por jogador e time.	`double`
`LancesLivresTentados`	Quantidade de arremessos de lances livre tentados por jogador e time.	`double`
`PorcentagemLancesLivres`	Percentual de arremessos de lances livre convertidos por jogador e time.	`double`
`RebotesOfensivos`	Quantidade de rebotes ofensivos por jogador e equipe.	`double`
`RebotesDefensivos`	Quantidade de rebotes defensivos por jogador e equipe.	`double`
`TotalRebotes`	Quantidade de rebotes totais por jogador e equipe.	`double`
`Assistencias`	Quantidade de assistências por jogador e equipe.	`double`
`RoubosBola`	Quantidade de roubos de bola por jogador e equipe.	`double`
`Tocos`	Quantidade de tocos por jogador e equipe.	`double`
`DesperdiciosBola`	Quantidade de desperdícios de bola por jogador e equipe.	`double`
`Faltas`	Quantidade de faltas por jogador e equipe.	`double`
`Pontos`	Quantidade de pontos por jogador e equipe.	`double`
`Idade`	Idade do jogador na temporada.	`Integer`
`SiglaTime`	Sigla do time em que o jogador participou a temporada.	`string`
`Jogos`	Quantidade de jogos em que o jogador atuou.	`Integer`
`JogosIniciados`	Quantidade de jogos em que o jogador iniciou a partida.	`Integer`

Fonte: Elaborado pelo autor.

Quadro 24 - Objeto `EstatisticaAvancada`

`EstatisticaAvancada` – objeto interno contendo as estatísticas avançadas do documento `JogadorEstatistica`.		
CAMPO	**DESCRIÇÃO**	**TIPO**
`Idade`	Idade do jogador na temporada.	`double`
`SiglaTime`	Sigla do time em que o jogador participou a temporada.	`double`
`Jogos`	Quantidade de jogos em que o jogador atuou.	`double`
`EficienciaJogador`	Percentual de eficiência dos arremessos.	`double`
`PorcentagemArremessosEficientes`	Percentual de eficiência dos arremessos convertidos.	`double`
`TaxaTentativas3Pontos`	Percentual de arremessos de 3 pontos comparados aos arremessos tentados pelo jogador.	`double`
`TaxaTentativasLancesLivres`	Percentual de lances livres comparados aos arremessos tentados pelo jogador.	`double`
`PorcentagemRebotesOfensivos`	Percentual de rebotes ofensivos enquanto está em quadra.	`double`
`PorcentagemRebotesDefensivos`	Percentual de rebotes defensivos enquanto está em quadra.	`double`
`PorcentagemRebotesTotal`	Percentual de rebotes totais enquanto está em quadra.	`double`
`PorcentagemAssistencias`	Percentual de assistências enquanto está em quadra.	`double`
`PorcentagemRoubosBola`	Percentual de roubos de bola enquanto está em quadra.	`double`
`PorcentagemTocos`	Percentual de roubos de tocos enquanto está em quadra.	`double`
`PorcentagemDesperdiciosBola`	Percentual de desperdícios de bola enquanto está em quadra.	`double`
`PorcentagemUsoJogador`	Estimativa de quanto o jogador é utilizado enquanto está em quadra.	`double`
`ContribuicaoVitoriaOfensiva`	Percentual de quanto o jogador contribui para o ataque do time.	`double`
`ContribuicaoVitoriaDefensiva`	Percentual de quanto o jogador contribui para a defesa do time.	`double`
`ContribuicaoVitoria`	Percentual do número de partidas o jogador contribui.	`double`
`EstimativaContribuicaoOfensiva`	Estimativa de ações ofensivas do time no período em que o jogador permanece em quadra.	`double`
`EstimativaContribuicaoDefensiva`	Estimativa de ações defensivas do time no período em que o jogador permanece em quadra.	`double`
`EstimativaContribuicaoTotal`	Estimativa de saldo de pontos do time no período em que o jogador permanece em quadra.	`double`

Fonte: Elaborado pelo autor.