INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DO RIO GRANDE DO SUL
CAMPUS RESTINGA
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DESEMPENHO DE AUTENTICAÇÃO EM APLICAÇÕES WEB ESCALÁVEIS NO ECOSSISTEMA BACK-END JAVASCRIPT

MATHIAS GHENO AZZOLINI

PORTO ALEGRE
2017

MATHIAS GHENO AZZOLINI

DESEMPENHO DE AUTENTICAÇÃO EM APLICAÇÕES WEB ESCALÁVEIS NO ECOSSISTEMA BACK-END JAVASCRIPT

Trabalho de Conclusão de Curso apresentada como requisito parcial para obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas da Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - IFRS, Campus Restinga.

Orientador: Régio Mechelin

Coorientador: Roben Lunardi

Porto Alegre

2017

MATHIAS GHENO AZZOLINI

DESEMPENHO DE AUTENTICAÇÃO EM APLICAÇÕES WEB ESCALÁVEIS NO ECOSSISTEMA BACK-END JAVASCRIPT

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - IFRS, Campus Restinga.

Data de Aprovação: 19/12/2017

Banca Examinadora

_____

Prof. Mestre Régio Antônio Michelin - IFRS - Campus Restinga
Orientador

_____

Prof. Mestre Roben Castagna Lunardi - IFRS - Campus Restinga
Coorientador

_____

Prof. Dr. Rafael Pereira Esteves - IFRS - Campus Restinga
Membro da Banca

_____

Prof. Mestre Diego Moreira da Rosa - IFRS - Campus Restinga
Membro da Banca

Um agradecimento especial para o meu orientador Régio Michelin, obrigado pela paciência e pelas dicas. Como também, para meu coorientador Roben Lunardi. Obrigado pela disponibilidade e sinceridade.

Dedico esse trabalho aos meus pais, sem eles nada disso seria possível. Obrigado por todas as noites em claro que dedicaram a mim.

Por fim, um muito obrigado a todos os meus amigos que aceitaram minha ausência prolongada durante o período de desenvolvimento desse trabalho.

Sometimes it is the people no one can imagine anything of who do the things no one can imagine.

Alan Turing

RESUMO

Este trabalho propõe o desenvolvimento de soluções web utilizando os três principais frameworks HTTP(S) disponíveis para o ecossistema JavaScript *Back-End* (Node.js). Para cada *Framework* serão implementadas duas estratégias de autenticação. Com base nos testes realizados e nos resultados obtidos foi observada uma vantagem significativa no uso da estratégia de autenticação *Session* implementado no framework *Express*.

Palavras-chave: Node.js, Desempenho, Autenticação

ABSTRACT

This paper proposes the development of three HTTP(S) Web solutions provided by frameworks availables on the NodeJS Platform and the implementation of two authentication's strategies for each framework. Based on the tests performed and the results obtained, a significant advantage was observed in the use of the Session authentication strategy implemented in the Express framework.

Keywords: Node.js, Performance, Authentication.

LISTA DE FIGURAS

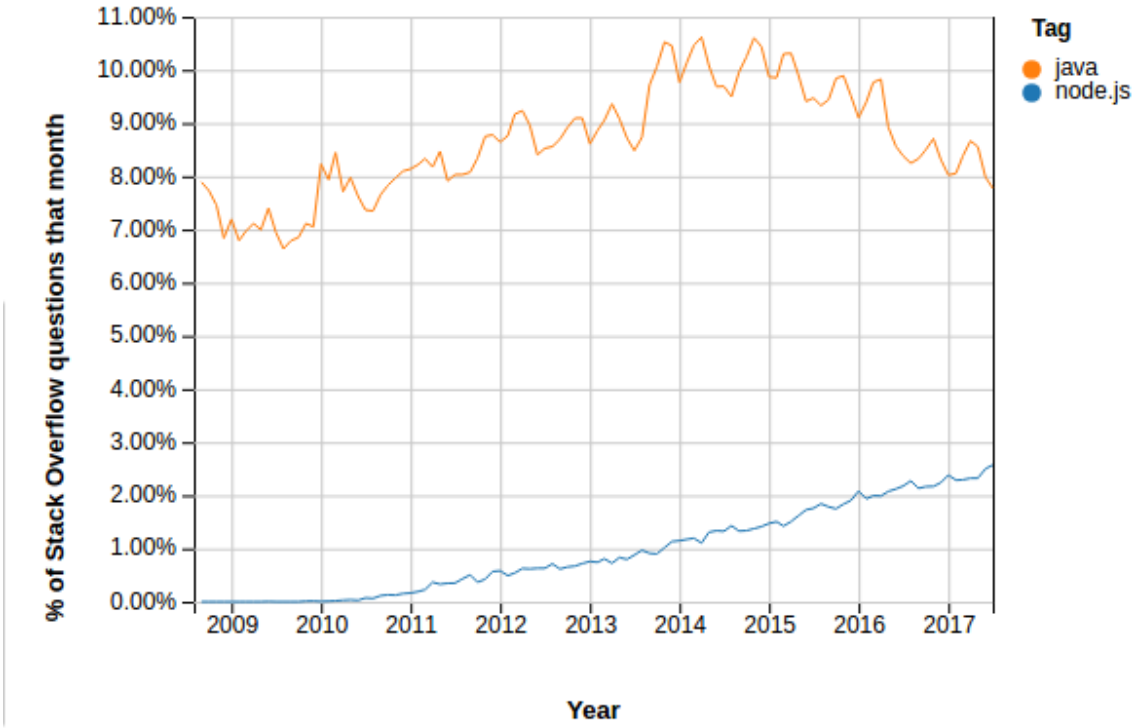
LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de sistemas Web modernos exigem a implementação de diversas ferramentas que auxiliam no sucesso da disponibilidade e funcionalidade do serviço. A fim de entregar uma boa experiência para os usuários, novas tecnologias são lançadas diariamente e a velocidade que essas tecnologias surgem no mercado impossibilita a análise de seu desempenho e das melhoras práticas de implementação. Node.js se enquadram nessa situação. O avanço da tecnologia possibilita novas ferramentas e frameworks a cada semana. Mikeal Rogers, criador da Node.JS Foundation, conta em um entrevista que a popularidade do Node.js ganhou proporções gigantes, podendo, inclusive, ultrapassar Java como a linguagem de desenvolvimento mais popular, considerando o crescimento da popularidade do Node.js e a diminuição do uso do Java (GIENOW, 2017) (Figura 1). Não apenas o Node.js cresceu, mas também seu gerenciador de pacotes - NPM (*Node.js Package Management*) que se tornou o maior gerenciador de pacotes do mundo, ultrapassando o Maven, do JAVA (DEBILL, 2017).

Figura 1 – Questões do Java e Node.js no StackOverflow



Fonte: (EXCHANGE, 2017)

Mesmo que muitas tecnologias existam elas podem não ser implementadas corretamente,

ou até mesmo podem não ser a solução ideal para o produto final. Existem diversos sites que possuem problemas de instabilidade constantes que impossibilitam o usuário de acessar a informação ou requisitar um serviço. Um exemplo quase anual disso no Brasil é o site do INEP que fornece cadastro de estudantes para a realização da prova do ENEM ou até mesmo para verificar o resultado final (BASSETTE, 2006). Esse tipo de situação não está apenas situado no Brasil, exemplos estrangeiros são comuns. Em 2009, vários servidores do mundo caíram ou sofreram instabilidades após a divulgação da morte do cantor Michael Jackson, Google e Twitter são exemplos de serviços que ficaram indisponíveis (CORDEIRO, 2006), (CAMARGO, 2010).

A Internet foi um marco na humanidade por proporcionar acesso fácil a uma quantidade muito grande de informações. Como James Gleick conta em *A Informação* (GLEICK, 2013), desde quando as máquinas foram capazes de exercer atividade intelectuais, como cálculos, os benefícios para a ciência e inovação foram enormes. O mundo vivencia um aumento expressivo da quantidade de informações digital gerada e processada, conforme explica. E para os próximos anos essa realidade deve ser garantida com o crescimento da internet, através da Internet das Coisas (*Internet of Things* - IoT) - que irá proporcionar até 2020 cerca de 50 Bilhões de dispositivos conectados na rede (BENAMAR ANTONIO JARA, 2013).

Por conta disso, fornecer acesso a milhares de usuários é uma realidade cada vez maior para todos os sites que estão hospedados no serviço de internet global. Entretanto, nem todos esses sites possuem tecnologias que garantem a Escalabilidade Horizontal (*Horizontal Scalability*) ou que fazem o balanceamento adequado do fluxo de dados - como a funcionalidade não-bloqueante do Node.js, que impossibilita o travamento de futuras requisições quando existe uma execução que exige muito recursos do servidor. Grandes empresas já estão saindo na frente e desenvolvendo soluções com tecnologias mais adequadas para situações que exigem muitos acessos de usuários. Netflix, Paypal (AVRAM, 2013) (HARRELL, 2013), eBay (PADMANABHAN, 2013) (RUPLEY, 2016), e Walmart são algumas das empresas que estão utilizando o Node.js em seus sistemas.

Entretanto, não apenas serviços públicos, onde todos podem acessar a informação sem se identificar, necessitam possuir tais qualidades técnicas. Existem diversos sistemas que fornecem acesso à informação apenas para usuários devidamente autenticados. Um exemplo popular são as redes sociais, como o twitter (TWITTER, 2017) e o facebook (FACEBOOK, 2017), que permitem acesso a algumas informações apenas para o utilitário da conta. Mas, além de redes sociais, sistemas Bancários, Educacionais, Hospitalares, Militares e Governamentais possuem um grande acervo de dados que apenas pessoas autorizadas devem acessar, além de fornecer

disponibilidade a centenas de milhares de requisições por dia ou por minuto. Em 2013 foi gerada uma grande discussão mundial em prol da privacidade de dados quando Edward Snowden revelou o esquema de espionagem da NSA (*National Security Agency*), provando a fragilidade de centenas de sistema ao fornecer dados sigilosos a pessoas não autorizadas (GRENWALD, 2013).

Com base nos problemas demonstrados anteriormente e considerando a importância de garantir instabilidade para centenas de milhares de requisições em sistemas que possuem dados sigilosos ou que não devem ser acessados por terceiros , este trabalho propõe a análise de desempenho de autenticação de sistema Web escaláveis baseados no ecossistema back-end javascript para os principais frameworks e estratégias de autenticação disponíveis para o ecossistema Node.js.

Para alcançar esse objetivo, este trabalho propõe: (i) analisar o desempenho de cada frameworks individualmente; (ii) verificar vantagens e desvantagem de cada estratégia de autenticação para cada framework; (iii) apresentar uma arquitetura seguindo padrões de projeto e boas práticas de desenvolvimento; por fim, (iv) realizar uma avaliação sugerindo a melhor solução para sistema escaláveis.

Este trabalho é estruturado da seguinte forma: 2) Referencial Teórico, onde são abordados os pontos teóricos fundamentais para o compreendimento da elaboração do trabalho. 3) Arquitetura Proposta, onde é explicada a arquitetura desenvolvida para a implementação dos testes; 4) Desenvolvimento, onde apresento as principais diferenças entre os projetos desenvolvidos. 5) Resultados, onde apresento os resultados obtidos e 5) Conclusão e Trabalhos Futuros, onde é feita a conclusão do trabalho e listadas as melhorias necessárias.

2 REFERENCIAL TEÓRICO

Neste capítulo serão discutido os principais pontos quanto a utilização do JavaScript no Back-End de uma aplicação, suas vantagens e o funcionamento interno do Node.js. Além disso, explicar o que significa o NoSQL *(Not Only SQL)* e o motivo pelo qual grandes empresa de tecnologias estarem utilizado esse tipo de solução em grande escala. Por fim, são apresentados os principais conceitos por trás da importância da privacidade de dados e sua importância para os sistema de grande porte.

2.1 JAVASCRIPT NO BACK-END

Javascript foi uma linguagem de desenvolvimento inicialmente criada para rodar pequenos programas dentro do navegador, desde sua criação, em 1995. Além disso, foi concebida para ser uma linguagem script interpretada multi-paradigma (Orientação a Objetos, Funcional), com tipagem dinâmica e funções de primeira classe, além de ser padronizada pelo padrão *ECMAScript*. Por conta dessas qualidades, JavaScript se tornou a principal linguagem de desenvolvimento front-end de aplicações web.

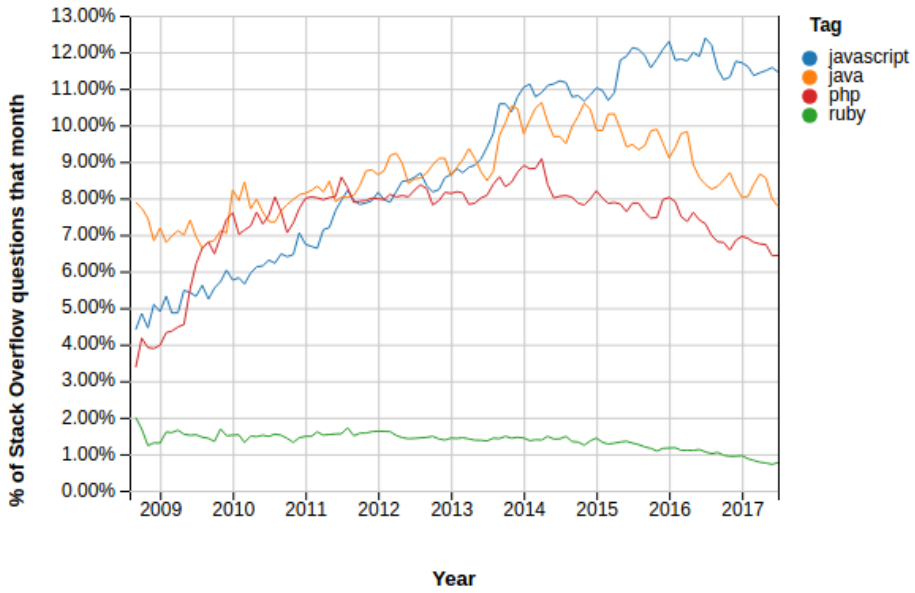
JavaScript também ganhou força em Banco de Dados que usam sua estrutura, ou estruturas muito semelhantes e derivadas, do JSON (*JavaScript Object Model*). JSON é "um subset derivado da linguagem javascript" que serve para representar e estruturar dados com um conjunto de propriedades que possibilitam a leitura humana. Sua estrutura é formada por uma lista de chave-valor, onde o valor pode ser *String*, *Number*, *Object*, *Array*, *boolean* ou *null* e a chave sempre uma *String*. Por conta da popularidade do JSON, muitos banco de dados Relacionais e NoSQL já suportam o JSON nativamente para transferência e armazenamento de dados.

O aprimoramento do JavaScript não se limitou apenas ao desenvolvimento de soluções front-end. Em 2009 Ryan Dahl criou o Node.js (DAHL, 2009b), uma plataforma que permite executar código JavaScript no *Back-End* que trouxe ao mundo do JavaScript inúmeros avanços. Node.js é uma plataforma escrita em C/C++, baseada na Engine V8, o interpretador JavaScript do Google Chrome, orientada a eventos e I/O (*Input and Output*) não bloqueante que utiliza o padrão CommonJS para seus módulos. Uma de suas principais diferenças entre as linguagens tradicionais de *Back-End* é a utilização de uma única thread, sendo assim *Single Thread*. Como Ryan Dahl conta em sua palestra de introdução ao Node.js (DAHL, 2009a), seu funcionamento interno é resultado da tentativa de melhorar a forma como as aplicações lidam com eventos de I/O , como o consumo de um serviço externo ou uma requisição a um banco de dados. Node.js

é uma tecnologia leve e eficiente ideal para criação de aplicações Web de acesso intensivo de dados, ganhando destaque entre as outras linguagem novas de desenvolvimento Web Python e PHP em testes de performance em requisições, cálculos e acesso ao banco de dados (LEI YINING MA, 2014).

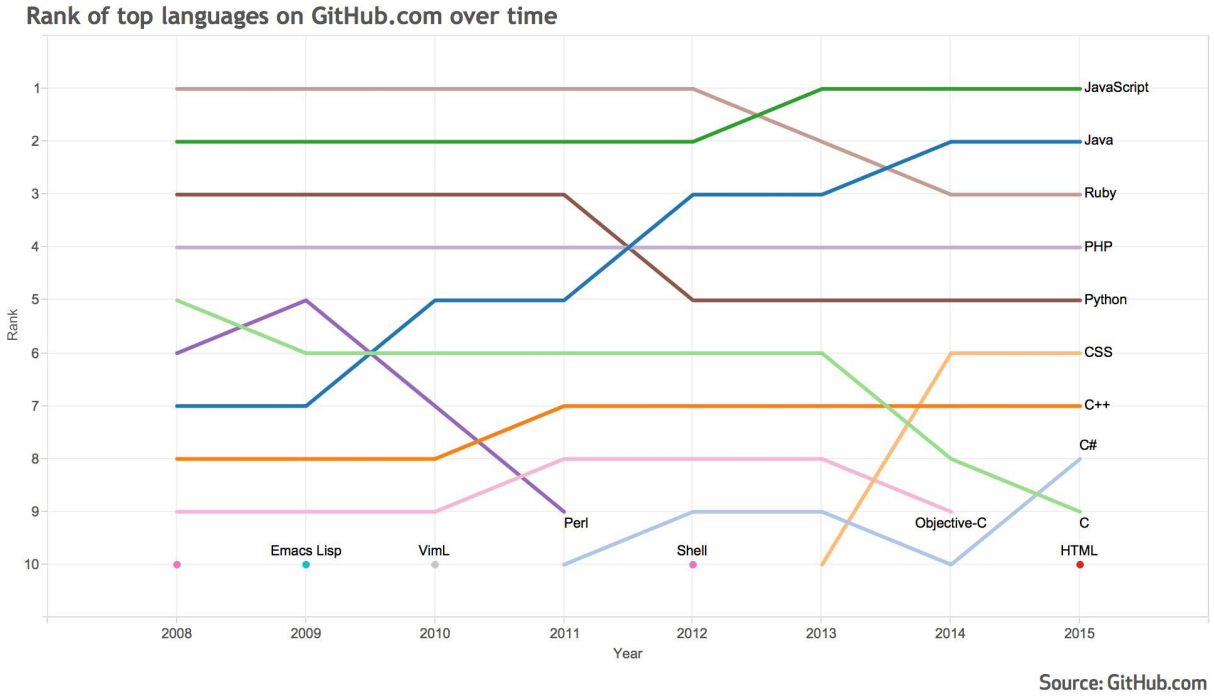
Por conta do domínio do JavaScript em todas as camadas de desenvolvimento de uma solução de software (*Back-End*, *Front-End*, e Banco de Dados) JavaScript se tornou a linguagem mais popular de desenvolvimento em tag no StackOverflow (Figura 2) e em repositórios no GitHub (Figura 3). Por conta desse domínio, o termo Full Stack foi designado a desenvolvedores que possuíam a capacidade de desenvolver produtos utilizando todas as Stacks de desenvolvimento. MEAN é o acrônimo designado para tecnologias JavaScript que juntas tornam possível o desenvolvimento *Full Stack* utilizando JavaScript como linguagem comum. "M" representa MongoDB, um banco de dados não-relacional (NoSQL), de alta performance e alta escalabilidade, orientado a documentos. "E" representa o Express.js um framework de desenvolvimento que facilita a criação de servidores HTTP(S). "A" representa o framework de desenvolvimento Front-End AngularJS, utilizada para criar páginas web SPA (*Single Page Applications*) . Por fim, temos o "N" que representa o Node.js, uma plataforma que possibilita a execução de JavaScript no *Back-End* de uma aplicação.

Figura 2 – Questões do Java, PHP, Ruby e JavaScript no StackOverflow



Fonte: (EXCHANGE, 2017)

Figura 3 – Rank Top Linguagens no GitHub



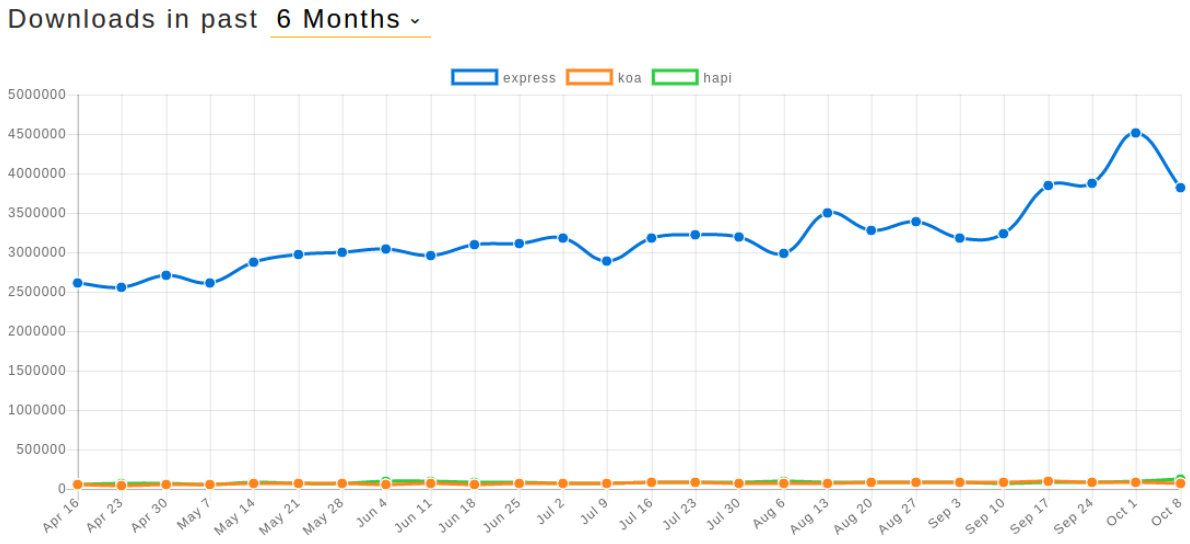
Fonte: (GITHUB, 2015)

Escalabilidade é uma funcionalidade necessária em qualquer empresa, especialmente em empresas mundiais ou sistema que são acessados por muitos usuários. Node.js se torna uma ótima opção para essas empresas e sistema, justamente por conta da facilidade em escalar. Walmart, por exemplo, possui cerca de 20000 requisições por segundo em seu site e aplicativo mobile em datas comemorativas, como o natal. Segundo Maxime Najim (FOUNDATION, 2017b), arquiteto sênior da Walmart, Node.js consegui entregar muito mais valor em menos linhas de código com o Assincronismo de I/O. NASA também utilizou dos recursos do Node.js para criar uma aplicação na nuvem que fornecesse dados concisos de uma forma eficiente através de APIs e microservices que rendeu um aprimoramento de mais de 300% (FOUNDATION, 2017a) .

No gerenciador de pacotes do Node.js, existe uma quantidade grande de frameworks para a criação de servidores HTTP(S). Dentro todos os mais popular é o Express.JS (ver figura 4), um dos primeiros no mercado - lançado em 2010. Entretanto, outros frameworks que possuem o mesma funcionalidade básica, criação de servidores HTTP(S,) se destacam ao introduzir novas abordagens de desenvolvimento, como o Koa.js (KOA, 2017) e o Hapi.js (HAPI, 2017). Conforme pode ser visualizado no gráfico da Figura 4, o Express.JS possui uma quantidade mensal de downloads cerca de 43 vezes maior que os outros dois frameworks , conforme pode

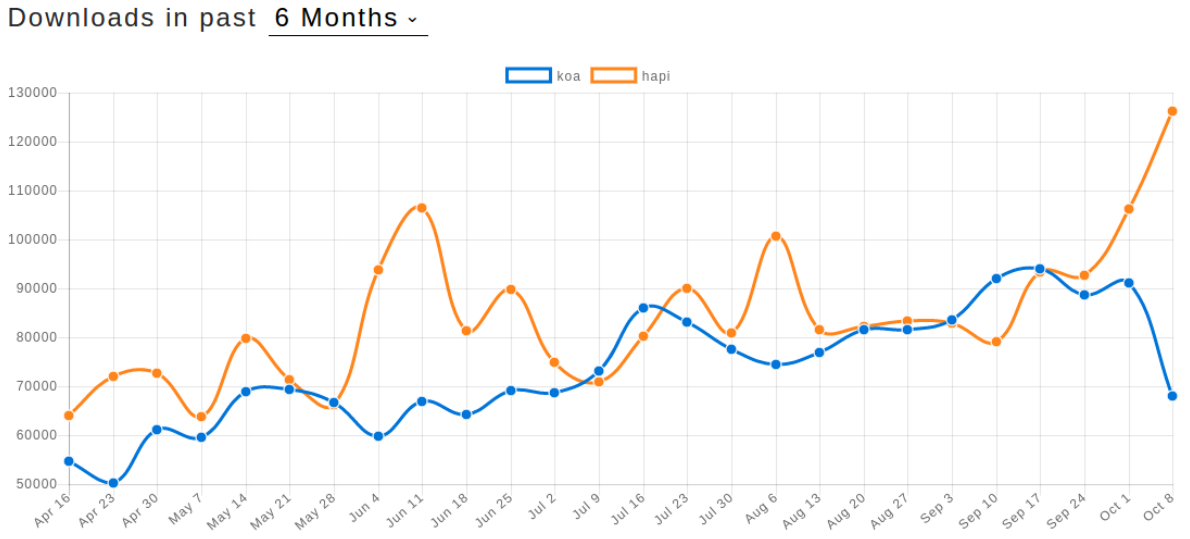
ser vista na Tabela 1, porém quando analisamos Koa.js e Hapi.js de forma isolada ao Express.js conseguimos ver que ambos são, também, muito utilizados (Figura 5).

Figura 4 – Express, Koa e Hapi Popularidade no NPM



Fonte: (NPM, 2017a)

Figura 5 – Koa e Hapi Popularidade no NPM



Fonte: (NPM, 2017b)

Tabela 1 – Média de Downloads Hapi, Express e Koa

	express	**koa**	**hapi**
10 dez	4.183.589,00	100.063,00	84.297,00
3 dez	4.020.190,00	98.694,00	91.627,00
26 nov	3.809.839,00	85.602,00	118.480,00
19 nov	4.267.451,00	100.227,00	92.544,00
média	4.070.267,25	96.146,50	96.737,00

Fonte: (NPM, 2017a)

2.2 NOSQL

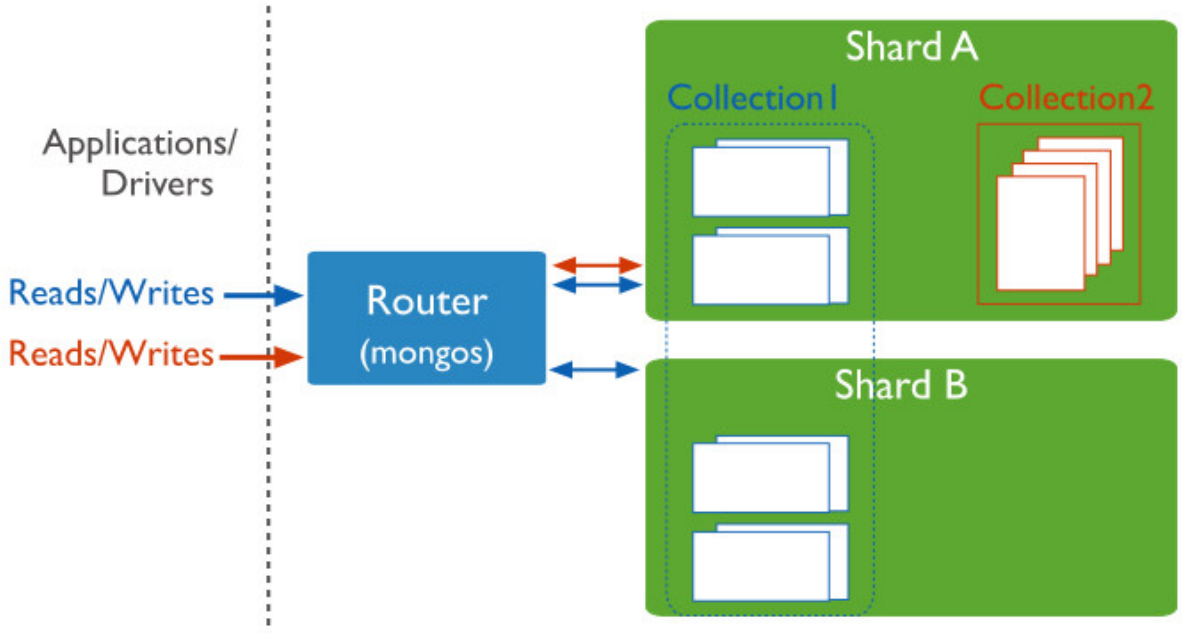
Afim de suportar uma quantidade muito grande de armazenamento de dados e garantir a escalabilidade da aplicação novas soluções em Banco de Dados alternativas aos SGBDs tradicionais surgiram, essas soluções são chamadas de NoSQL. Os bancos de dados NoSQL atualmente podem ser classificados em 5 tipos principais: (i) Armazenamento de Chave/Valor; (ii) Armazenamento de Super Colunas; (iii) Armazenamento de Documentos; (iv) Armazenamento de Grafos e (v) Armazenamento Orientado Objetos (ALEXANDRE, 2013).

Os banco de dados tradicionais disponibilizam um conjunto de funcionalidades que é explicado pela termo ACID, o acrônimo de *Atomicity*, *Consistency*, *Isolation*, *Durability*. Essas funcionalidades possibilitam um sistema SGBD tradicional e possuir uma baixa escalabilidade e uma alta consistência de dados. Já as tecnologias NoSQL não seguem esse padrão de funcionalidades, o termo utilizado para tecnologias NoSQL é o BASE, acrônimo de *Basically Avaliable*, *Soft state*, *Eventual Consistency*. Diferente do ACID, o BASE oferece alta disponibilidade e baixa consistência de dados, justamente o que grande empresas de tecnologias buscam (BRITO, 2010). Existe uma quantidade grande de softwares disponíveis que oferecem os serviços não relacionais, tais como Redis, Cassandra, CouchDB, MongoDB, BigTable, RethinkDB, DymamoDB, HBase, entre outros - muitos desses bancos de dados NoSQL são mantidos por empresas como Google e Facebook. Facebook, por exemplo, já utilizou o banco de dados Cassandra em sua arquitetura. Esse banco de dados NoSQL, conforme (MARROQUIM, 2012) explica, possui uma arquitetura distribuída baseada no DynamoDB que tem um funcionamento semelhante ao protocolo P2P (*Peer-To-Peer*) - O que garante distribuição de dados em escala global. Dentre todas as excelentes soluções NoSQL o MongoDB se tornou o mais popular por conta de diversas funcionalidades que facilitam o seu uso, tais como o *Sharding* que possibilidade a facilidade ao escalar horizontalmente uma aplicação e o *Replica Set* que garantir redundância e alta disponibilidade de forma simples e com uma curva de aprendizado baixa.

MongoDB é um banco de dados orientado a documentos. Todos os documentos são armazenados em uma collection e uma ou várias dessas são armazenadas em um banco de dados. Uma instância do MongoDB pode conter centenas de banco de dados. A menor unidade disponível é o documento, que é estruturado de uma forma muito semelhante ao JSON. Além disso, o MongoDB não exige autenticação para registro de dados, como também não possui Schema, sendo assim não existe uma padronização para a adição de documentos em uma coleção: todos os documentos são aceitos, por conta disso o controle e a validação do Input dos documentos deve ser feito através da aplicação. Por conta disso, o MongoDB facilita bastante o *Input* de dados, obtendo vantagens na performance quando comparado com outros banco de dados. Conforme (POLITOWSKI, 2013) mostra o MongoDB obtem grande vantagens quando comparado com um SGBD PostgreSQL na inserção de dados, chegando a ser dez vezes mais rápido em inserções complexas. Essa vantagem não é obtida apenas quando comparado com o PostgreSQL, em uma comparação entre MySQL e MongoDB o MongoDB também obtem resultados muito melhores que o MySQL nas operações CRUD (Create, Read, Update, Delete) chegando a ser: mil vezes mais rápido em Insert, 120 vezes mais rápido em Select, 300 vezes mais rápido em Update e 120 vezes mais rápido em operações Delete (GYőRöDI ROBERT GYőRöDI, 2015).

Sharding é a principal funcionalidade do MongoDB que garante a Escalabilidade Horizontal da aplicação. Escalabilidade Horizontal é o termo genérico não pertencente apenas ao mundo dos banco de dados, esse termo é designado a sistema que distribuem seu poder de armazenamento ou processamento entre várias máquinas, sendo assim, não centralizado. A principal vantagem de um sistema que possui Escalabilidade Horizontal é garantir o crescimento computacional da aplicação. Sendo assim, *Sharding* é um método que disponibiliza o armazenamento de dados em múltiplos dispositivos rodando instâncias do MongoDb em *Sharding* - Chamado de Mongos, que em geral somam um único serviço de armazenamento. Na figura 6 podemos ver um exemplo de *Sharding* com uma *collection* do MongoDB.

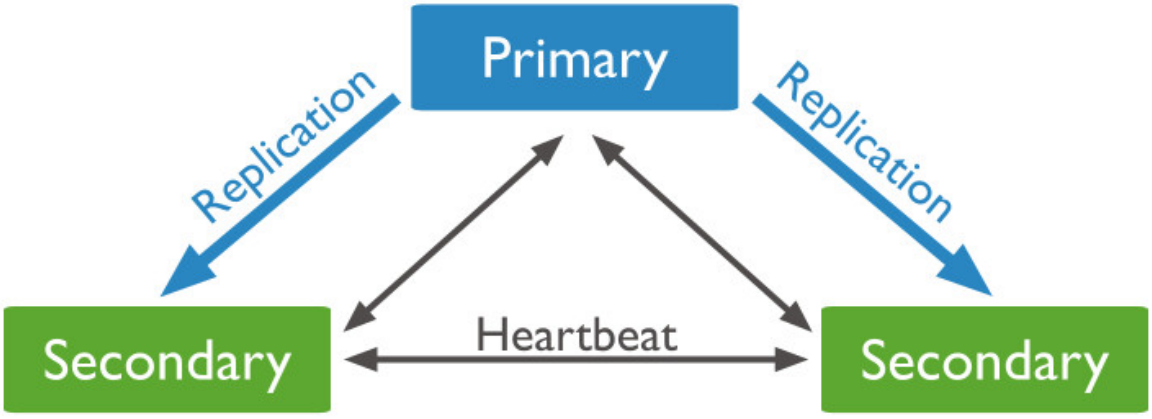
Figura 6 – Funcionamento do Sharding



Fonte: (MONGODB, 2017b)

Já o *Replica Set* é a principal ferramenta do MongoDB que garante a redundância de dados, sendo uma prática fundamental para *Deploy* da aplicação. No MongoDB um Replica Set é a execução de instâncias do MongoDB que funcionam interligados entre si, sendo o serviço primário responsável por leitura e gravação de dados, e o serviço secundário uma cópia exata do dados do serviço primário. Existem duas formas de arquitetura do *Replica Se*t: (i) um primário e dois secundários, (ii) um primário, um arbitrário e um secundário. Veja na figura 7 um exemplo de uma arquitetura que possui dois secundários e um primário.

Figura 7 – Funcionamento do Replica Set

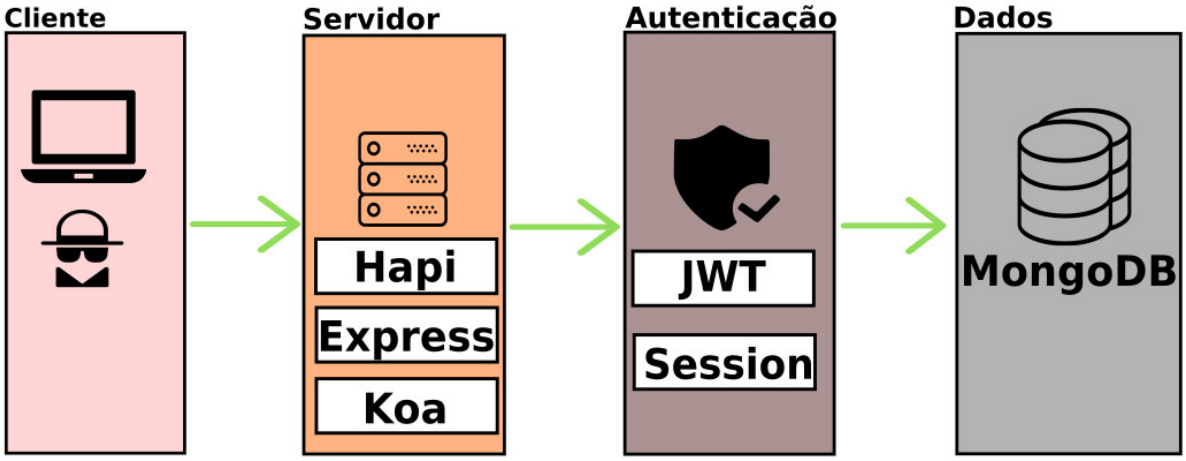


Fonte: (MONGODB, 2017a)

3 ARQUITETURA PROPOSTA

Para realizar os testes necessários e com o objetivo de promover semelhanças entre os código de todos os servidores desenvolvidos, será utilizada um padrão de arquitetura para todos os frameworks. O fluxo básico das aplicações é dividida em três partes fundamentais: (i) Servidor, o qual atende requisições recebidas através de um canal criptografado usando HTTPS, (ii) Sistema de Autenticação, que implementa a estratégia de autenticação e (iii) Acesso ao Banco de dados, de onde são armazenadas e lidas os dados. A única camada que não será analisada será o *Front-End*, a camada que faz requisições ao servidor.

Figura 8 – Ilustração da Arquitetura



O servidores HTTP(S) possuem uma camada de segurança para o acesso ao servidor, o TLS (*Transport Layer Security*). Por ser uma boa prática, garantir segurança de tráfego de dados e por ser uma tecnologia indispensável em aplicações em homologação, o uso do TLS é fundamental para a aplicação. Além disso, será adotado o padrão de arquitetura API REST (*Representational State Transfer*), que permite a transferência de dados entre aplicações através do protocolo HTTP (FIELDING, 2009).

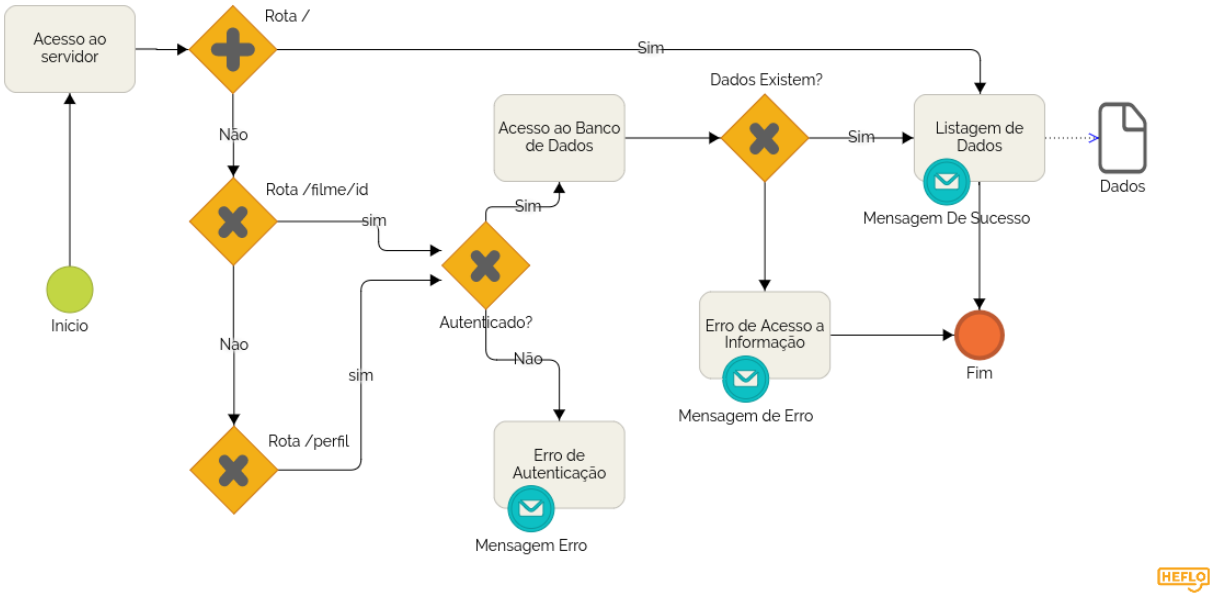
Todos os servidores terão rotas de acesso pré-definidas. No total serão 4 rotas padrões implementadas em todos os servidores - três dessas rotas serão usadas para testar o desempenho do sistema e uma para realização da autenticação. Todas as rotas utilizadas irão acessar o banco de dados e das três rotas que serão utilizadas para medir o desempenho do sistema, duas são autenticadas. A tabela 2 lista em detalhes as diferenças de cada rota, e a imagem 9 ilustra o fluxo de autenticação para as três rotas que irão ser usadas para medir desempenho. O rota principal é a rota que exige mais do servidor, pois faz uso intensivo do banco de dados. Essa rota entrega

Tabela 2 – Proteção das Rotas

Rota	Permissão	Acesso ao Banco	Operação DB	Autenticação
/	Acessível Para Todos	Sim	Leitura	Não
/filme/id	Acessível Para Autenticados	Sim	Leitura	Sim
/perfil	Acessível Para Autenticados	Sim	Leitura	Sim
/login	Acessível Para Todos	Sim	Leitura	Não

para apenas uma requisição 100 registros do banco de dados. A rota que menos exige do servidor é a rota de perfil, que retorna poucos dados após o sucesso da autenticação.

Figura 9 – Fluxo das Rotas Utilizadas



Para a autenticação das Rotas do sistema serão utilizados duas estratégias de autenticação, o JWT e o Session. O JWT é uma tecnologias relativamente nova que ganhou muito destaque quando a Stack MEAN ganhou popularidade, já o Session é uma abordagem relativamente antiga, pois possui implementação para as diversas ferramentas tradicionais de criação de aplicações web autenticadas.

JWT é o acrônimo de *JSON Web Token* e sua funcionalidade básica é fornecer dados JSON através de uma *hash* criptografada, é padronizada pelo RFC 7519. Toda Hash JWT é dividida em três partes, um *Header*, *Payload* e o *Verify Signature*, separados por um ponto. No Header é onde colocamos as informações básicas para configurar o JWT, como a criptografia, no Payload é onde é colocado os dados em JSON que serão transportados para o outro destinatário, já o *Verify Signature* é o local onde conterá a chave criptográfica que servirá para validar a assinatura da *Hash*. Na figura 10 podemos ver um exemplo de *hash* JWT.

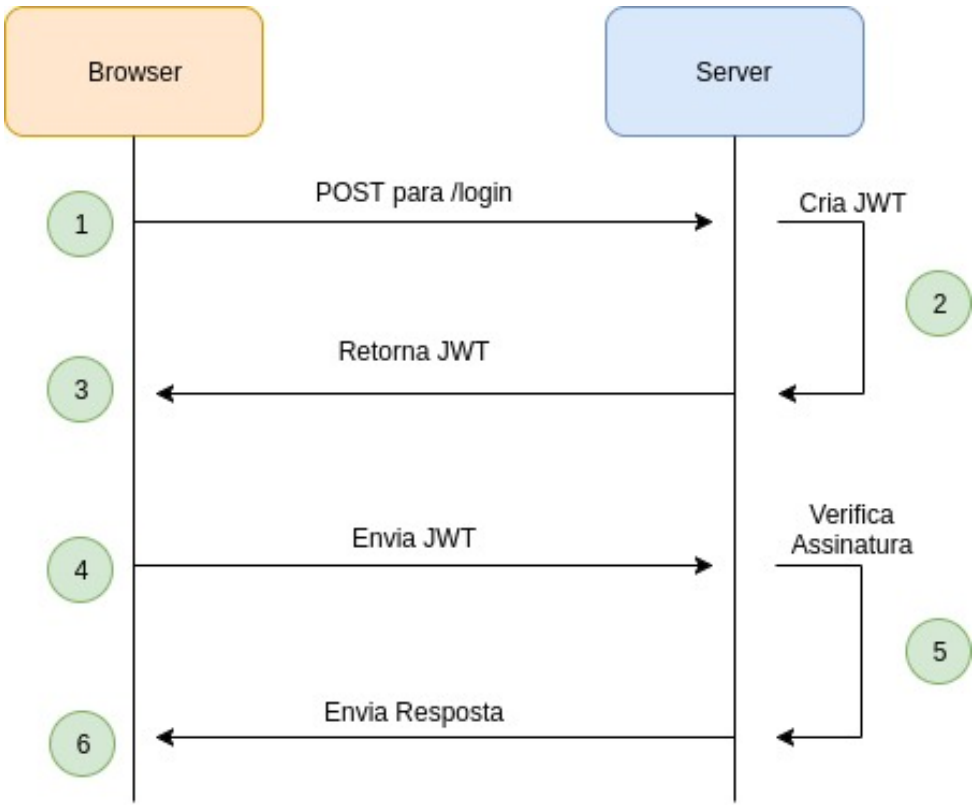
Figura 10 – Exemplo de Hash JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Fonte: (AUTH0, 2017)

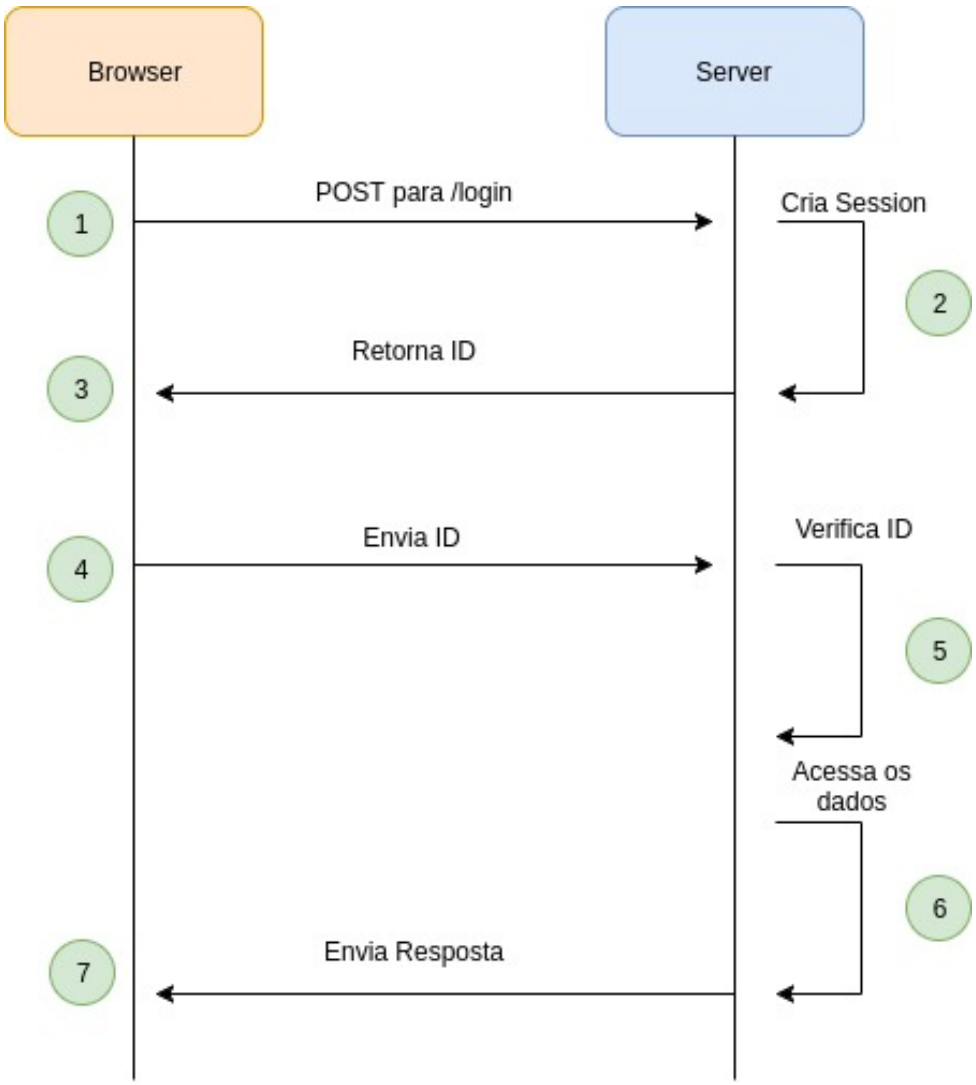
O uso do JWT é uma mudança no fluxo tradicional de um sistema de autenticação. Com o JWT é dado ao usuário a responsabilidade de entregar a *hash* para todas as próximas requisições ao servidor. Se o usuário não tiver a *hash* e tentar acessar uma rota protegida, ele não conseguirá acessá-la. Para um usuário conseguir gerar sua *hash* é necessário enviar através de uma requisição POST HTTP(S) o login e a senha do usuário para uma rota específica. O servidor irá gerar o *hash* e retorná-la para o usuário se os dados de login estiverem corretos. O usuário tendo posse dessa *hash* poderá fazer requisições autenticadas ao servidor. O fluxo pode ser visualizado na figura 11.

Figura 11 – Fluxo JWT



Diferente do JWT, aplicações que utilizam o *Session* como estratégia principal de autenticação não armazenam os dados em uma estrutura e a enviam para o usuário. No *Session*, os dados são deixados no servidor. Quando um usuário é autenticado um ID é gerado e entregue para o cliente, esse ID normalmente é armazenado no navegador em *Cookies*, por exemplo. Quando esse usuário faz uma requisição o servidor terá que procurar pelo *Session* do usuário no servidor através do ID fornecido. Se uma aplicação possui centenas de usuário, por exemplo, o servidor terá que gerar centenas de *Sessions* e armazená-las no lado do servidor. Ver figura 12.

Figura 12 – Fluxo Session



As duas estratégias de autenticação possuem vantagens e desvantagens. O JWT consegue reduzir o número de passos que o servidor deverá fazer ao autenticar o usuário, mas ao mesmo tempo, terá que utilizar mais recursos do servidor na hora de gerar a *hash* por conta de criptografia envolvida, além disso, todos os dados serão trafegados para o usuário. Já no *Session*, todas as *Sessions* teriam que ser armazenadas, atualizadas e acessadas a cada requisição de um usuário. A tabela 3 detalha algumas diferenças entre as principais estratégias.

Tabela 3 – JWT e Session Verificação

Estratégia	Dados	Verificação
JWT	Disponível na Hash	No servidor, com a Hash
Session	Disponível no Servidor	No servidor, com o ID

4 DESENVOLVIMENTO

Nesse capítulo serão apresentados os detalhes sobre o desenvolvimento das soluções de servidores HTTP(S) para os três frameworks utilizados. Além disso, serão detalhados os módulos utilizados para o desenvolvimento da solução agrupados, respectivamente, por estrategia de autenticação. Por fim, serão apresentados os detalhes do desenvolvimento da solução de teste de desempenho, os módulos, tecnologias e lógica utilizada. O código gerado foi armazenado no GitHub (AZZOLINI, 2017).

4.1 EXPRESS

Por ser o principal framework Node.js para soluções HTTP(S) o desenvolvimento foi bastante fluído. Documentação e plugins compatíveis com o Express são algumas das grandes vantagens do Express em comparação com os demais frameworks. Como pode pode ser visto na Tabela 4 foram utilizados 4 módulos para o desenvolvimento da aplicação que utiliza a estratégia de autenticação JWT. Já para a estratégia Session, conforme mostrado na Tabela 5, foram utilizados 6 módulos. Na Tabela 5 pode-se notar o uso do módulo "connect-mongodb-session". Esse módulo é uma recomendação de uso do plugin principal de autenticação session, o "express-session". Seu uso é recomendado para o melhor gerenciamento de memória para a persistência de objetos "sessions".

Tabela 4 – Dependências Express JWT

Dependência	Versão	Descrição
Express	4.15.3	Servidor HTTP(S)
body-parser	1.17.2	Parseamento JSON do body HTTP(S)
jsonwebtoken	7.4.1	Estratégia JWT
mongodb	2.2.31	Drive Oficial MongoDB

Tabela 5 – Dependências Express Session

Dependência	Versão	Descrição
Express	4.15.3	Servidor HTTP(S)
cookie-parser	1.4.3	Parseamento JSON do Cookie
body-parser	1.17.2	Parseamento JSON do body HTTP(S)
express-session	1.15.5	Estratégia Session para Express
connect-mongodb-session	1.3.0	Armazenamento de Session no MongoDB
mongodb	2.2.31	Drive Oficial MongoDB

4.2 HAPI

Hapi é um framework que presa pela facilidade de desenvolvimento. Seu estilo de desenvolvimento é focado na criação de estruturas semelhante ao JSON. Com isso, criar rotas, adicionar plugins, manipular o fluxo do usuário se torna uma tarefa simples e eficiente. Para o funcionamento das estratégias de autenticação no Hapi bastou-se utilizar os plugins disponíveis para o framework. No caso da estratégia de autenticação Session, conforme mostrado na Tabela 6, bastou utilizar o plugin "hapi-server-session", qua facilita a implementação. Para o JWT ocorreu o mesmo processo: Adição e configuração do plugins. Nesse caso, como mostrado na Tabela 7, o plugin utilizado foi o "hapi-auth-jwt2".

Tabela 6 – Dependências Hapi Session

Dependência	Versão	Descrição
hapi	16.6.0	Servidor HTTP(S)
hapi-server-session	3.0.2	Estratégia Session para Hapi
mongodb	2.2.31	Drive Oficial MongoDB

Tabela 7 – Dependências Hapi JWT

Dependência	Versão	Descrição
hapi	16.6.0	Servidor HTTP(S)
hapi-auth-jwt2	7.3.0	Estratégia JWT para Hapi
mongodb	2.2.31	Drive Oficial MongoDB

4.3 KOA

Por ser um framework que possui muitas semelhanças com o Express, desenvolver com o Koa não foi uma mudança grande de estilo de desenvolvimento quando consideramos a estrutura do framework. Porém, o koa têm como objetivo trazer as novidades do desenvolvimento javascript para o desenvolvimento de servidores HTTPs. Por conta disso foi necessário o entendimento e o desenvolvimento seguindo o padrão *assync / await* - uma novidade do *ECMAScript* 7. Para o desenvolvimento da estratégia de Session para o koa bastou-se utilizar o plugin "koa-session", conforme Tabela 8. Já para a estratégia JWT foi utilizada a biblioteca padrão do JWT o "jsonwebtoken"e o seu controle e verificação foi feito sem a utilização de plugins (Ver Tabela 9).

Tabela 8 – Dependências Koa Session

Dependência	Versão	Descrição
koa-session	5.5.0	Estratégia Session para Koa
koa	2.3.0	Servidor HTTP(S)
koa-body	2.3.0	Parse JSON para body HTTP(S)
mongodb	2.2.31	Drive Oficial MongoDB

Tabela 9 – Dependências Koa JWT

Dependência	Versão	Descrição
jsonwebtoken	5.5.0	Estratégia JWT
koa	2.3.0	Servidor HTTP(S)
koa-body	2.3.0	Parse JSON para body HTTP(S)
mongodb	2.2.31	Drive Official MongoDB

4.4 TESTES COM K6 E PERFORMANCE HOOKS

Para a realização dos testes foi utilizado, em conjunto, duas tecnologias: *Performance Hooks* e k6. O *Performance Hooks* é um módulo nativo do Node.js que foi adicionado na versão 8.5 da plataforma. Esse módulo tem como objetivo criar métricas de performance seguindo a especificação W3C Performance Timeline. k6 é uma ferramenta de teste de carregamento escrita em Go e JavaScript e tem como objetivo facilitar a criação de *Scripts* de testes de carregamento semelhante ao código de teste unitário.

O k6 foi utilizado para criação de *Script* de acesso as rotas descritas na tabela 2. Para garantir a consistência dos resultados o *Script* desenvolvido para o k6 garante a aleatoriedade do(a) (i) Rota Acessada, (ii) ID do Filme Acessado e (iii) Usuário Autenticado. O funcionamento interno do k6 permite a execução do *Script* com mais de um Virtual User (*Virtual User*) acessando o site, assim como o tempo em segundos que o Script será executado. k6 possui um método principal: default. Esse método fica rodando em loop infinito enquanto o tempo não expira, já os métodos e variáveis que não estão dentro no método default irão executar apenas uma vez por VU. Sendo assim, o método default representa as ações de um VU.

O *Performance Hooks* foi utilizado para fazer a medição da execução de cada rota do sistema. Para isso, foi criado uma classificação de tipo de métricas. Para cada execução bem sucedida da rota foi persistido no banco de dados um objeto JSON contendo informações pertinentes para a métrica. Na tabela 10 são descritos os campos desse objeto. Já na tabela 11 são descritos as classificações das métricas. Por fim, foi utilizado o *aggregate framework* do

MongoDB para realização de querys no banco de dados que retornassem os valores e as médias que são apresentadas no capitulo de Resultados.

Tabela 10 – Estrutura do objeto

Chave	Tipo de Dado	Descrição
_id	Date	Identificador da execução
duration	Double	Métrica de tempo de execução
startTime	Double	Métrica de início da execução
entryTyte	String	Tipo de métrica
name	String	Classificação da métrica pela rota
tipo	Number	Controle de execução de teste

Tabela 11 – Classificação dos objetos persistidos

Classificação	Descrição
FILMEID_EXPRESS_JWT	Rota Filme ID para Express JWT
FILMEID_EXPRESS_SESSION	Rota Filme ID para Express Session
FILMEID_HAPI_JWT	Rota Filme ID para Hapi JWT
FILMEID_HAPI_SESSION	Rota Filme ID para Hapi Session
FILMEID_KOA_JWT	Rota Filme ID para Koa JWT
FILMEID_KOA_SESSION	Rota Filme ID para Koa Session
HOME_EXPRESS_JWT	Rota Home para Express JWT
HOME_EXPRESS_SESSION	Rota Home para Express Session
HOME_HAPI_JWT	Rota Home para Hapi JWT
HOME_HAPI_SESSION	Rota Home para Hapi Session
HOME_KOA_JWT	Rota Home para Koa JWT
HOME_KOA_SESSION	Rota Home para Koa Session
LOGIN_EXPRESS_JWT	Rota Login para Express JWT
LOGIN_EXPRESS_SESSION	Rota Login para Express Session
LOGIN_HAPI_JWT	Rota Login para Hapi JWT
LOGIN_HAPI_SESSION	Rota Login para Hapi Session
LOGIN_KOA_JWT	Rota Login para Koa JWT
LOGIN_KOA_SESSION	Rota Login para Koa Session
PERFIL_EXPRESS_JWT	Rota Perfil para Express JWT
PERFIL_EXPRESS_SESSION	Rota Perfil para Express Session
PERFIL_HAPI_JWT	Rota Perfil para Hapi JWT
PERFIL_HAPI_SESSION	Rota Perfil para Hapi Session
PERFIL_KOA_JWT	Rota Perfil para Koa JWT
PERFIL_KOA_SESSION	Rota Perfil para Koa Session

5 RESULTADOS

Nesse capítulo serão apresentados os dados obtidos através da execução dos testes desenvolvidos. Os testes realizados geraram dados quanto a eficiência da execução bem sucedida de uma rota da aplicação, utilizando, na maioria dos casos, a estratégia de autenticação Session ou JSON Web Token (JWT). A fim de obter dados mais confiáveis, os testes foram rodados em duas etapas distintas e o resultados apresentados são a média do tempo de execução. Os testes foram executados em um ambiente de *Cloud* VULTR, utilizando VPS (*Virtual Private Servers*). Na Tabela 12 são apresentadas as especificações desses servidores.

Tabela 12 – Especificações VPS

Servidor	Memória	HD	CPU	País	SO
servidor Node.js	512 mb	25 Gb SSD	1	EUA	Ubuntu 16.04 LTS 64Bit
servidor MongoDB	1024 mb	25 Gb SSD	1	Japão	Ubuntu 16.04 LTS 64Bit
servidor MongoDB	1024 mb	25 Gb SSD	1	Alemanha	Ubuntu 16.04 LTS 64Bit
servidor MongoDB	1024 mb	25 Gb SSD	1	EUA	Ubuntu 16.04 LTS 64Bit

5.1 PRIMEIRO CASO DE TESTE

Na primeiro caso de testes foram utilizados tais parâmetros: (i) 1 VU, (ii) 10 segundos de execução e (iii) 5 repetições. A quantidade de vezes que cada rota foi acessada pode ser vista na Tabela 13. A média resultante do tempo de execução de cada rota pode visto na Tabela 14, já a representação visual desses dados podem ser vistos na Figura 13. O comparativo da média de execução de todas as rotas testadas para uma estratégia de autenticação pode ser visto na Tabela 15, assim como sua representação visual na Figura 14

Tabela 13 – Quantidade de Acessos

Rota	Express		Hapi		Koa	
	JWT	Session	JWT	Session	JWT	Session
/	22	19	24	13	19	18
/filme/:id	19	17	19	19	17	21
/perfil	18	14	12	22	23	18
/login	5	5	5	5	4	5

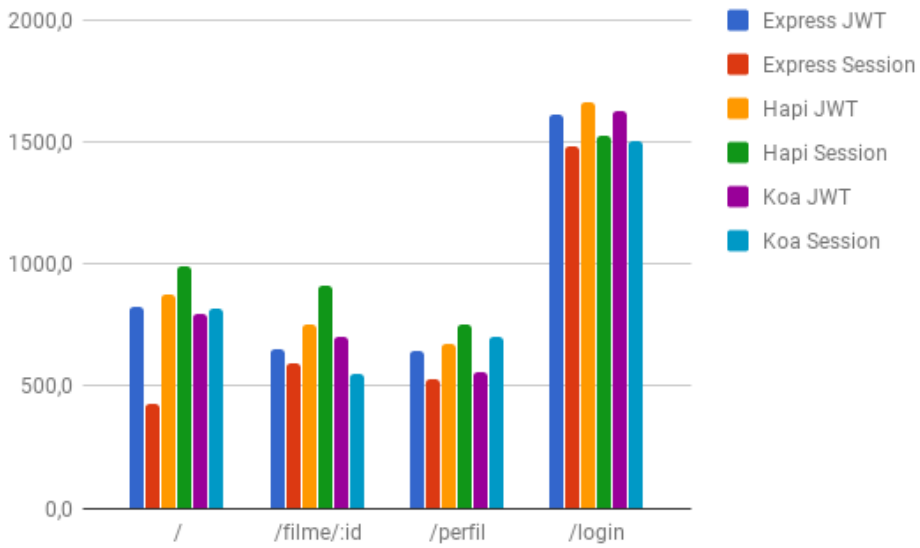
Na Tabela 14 podem ser vistas as métricas em milissegundo da média de execução das rotas testadas, agrupadas, respectivamente, por framework e estratégia de autenticação. Para a rota principal percebe-se que o melhor resultado é obtido pelo Express.js *Session*, que chega a ser 1.85 vezes mais rápido que o segundo mais rápido, e até 2.3 vezes mais rápido que o último

colocado. Para a rota de acesso aos detalhes de um filme o melhor resultado foi obtido pelo koa *Session*, porém a diferença entre o segundo mais rápido é baixa, sendo apenas 1.07 vezes mais rápido, já para o último colocado chega a ser 1.6 vezes mais rápido. Para a rota de perfil, o melhor resultado ficou com Express.js *Session* que, comparado com o segundo mais rápido não apresenta resultados tão performáticos, chegando a ser 1.05 vezes mais rápido que o segundo colocado e 1.4 vezes mais rápido que o último colocado. A rota login foi a que apresentou resultados mais lineares, sendo o mais rápido o Express.js *Session*, porém a diferença entre o segundo mais rápido e mais lento, são, respectivamente, 1.01 e 1.12 vezes mais rápido.

Tabela 14 – Duração Média em ms de execução da Rota

Rota	**Express**		**Hapi**		**Koa**	
	JWT	**Session**	**JWT**	**Session**	**JWT**	**Session**
/	827,4	426,9	874,4	989,7	793,3	819,1
/filme/:id	653,9	596,9	753,2	910,0	699,1	552,7
/perfil	644,3	528,7	669,6	750,5	558,6	704,6
/login	1613,9	1481,2	1663,2	1526,6	1624,0	1506,7

Figura 13 – Gráfico Duração Média (ms)

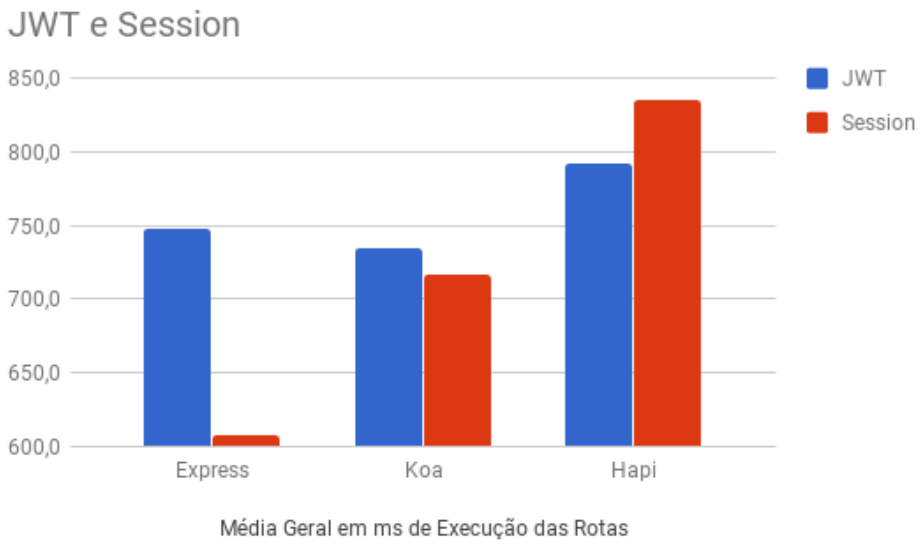


Percebe-se que o Express.js *Session* conseguiu se destacar nos resultados das rotas testadas, sendo o mais performático em velocidade de 3 de um total de 4 rotas. No resultado final da média de execução de todas as rotas para uma estratégia de autenticação o resultado não foi diferente: Express.js *Session* obteve o melhor resultado entre todos. O segundo resultado mais rápido também ficou com a estratégia *Session*, para koa.js. Express.js *Session* obteve um resultado 1.18 vezes mais rápido que o segundo colocado e 1.37 vezes mais rápido que o mais lento.

Tabela 15 – Média Geral em ms de Execução das Rotas

	JWT	Session
Express	747,9	606,7
Koa	735,0	716,6
Hapi	792,1	835,4

Figura 14 – Média Geral (ms)



5.2 SEGUNDO CASO DE TESTE

Para o segundo caso de testes foram utilizados tais parâmetros: (i) 5 VU, (ii) 10 segundos de execução e (iii) 10 repetições. A quantidade de vezes que cada rota foi acessada pode ser vista na Tabela 13. A média resultante do tempo de execução de cada rota pode visto na Tabela 17, já a representação visual desses dados podem ser vistos na Figura 15. O comparativo da média de execução de todas as rotas testadas para uma estratégia de autenticação pode ser visto na Tabela 18, e por fim a representação visual dessa média na Figura 16

Tabela 16 – Quantidade de Acessos

Rota	Express		Hapi		Koa	
	JWT	Session	JWT	Session	JWT	Session
/	59	37	41	38	60	58
/filme/:id	54	34	48	39	41	46
/perfil	44	45	61	48	51	46
/login	50	50	49	50	49	50

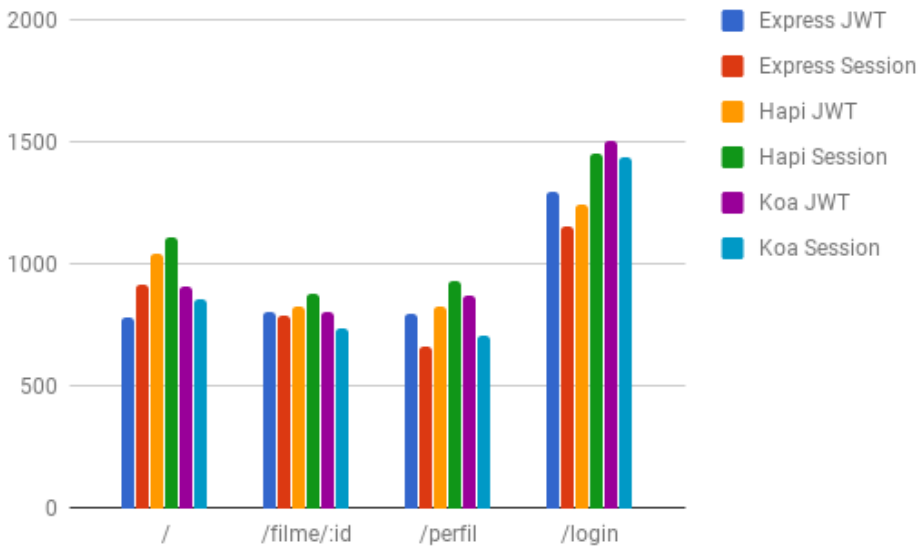
Na Tabela 17 podem ser vistas as métricas em milissegundo da média de execução das rotas testadas para o segundo caso de teste, agrupadas, respectivamente, por framework e

estratégia de autenticação. Para a rota principal percebe-se que o melhor resultado é obtido pelo express JSON Web Token, que chega a ser 1.09 vezes mais rápido que o segundo colocado, e até 1.4 vezes mais rápido que o último. Percebe-se que o resultado obtido aqui é bem divergente que o resultado obtido no primeiro caso de teste. Para a rota de acesso aos detalhes de um filme o melhor resultado foi obtido pelo koa session, porém a diferença entre o segundo mais rápido é baixa, sendo apenas 1.07 vezes mais rápido, ja para o último colocado chega a ser 1.19 vezes mais rápido, resultados semelhantes ao primeiro caso de teste. Para a rota de perfil, o melhor resultado ficou com express session que, comparado com o segundo mais rápido não apresenta resultados tão performáticos, chegando a ser 1.06 vezes mais rápido que o segundo colocado e 1.4 vezes mais rápido que o último colocado, sendo considerado um resultado quase semelhante ao apresentado no primeiro caso de teste. Já a rota de login para o segundo caso de teste também apresentou resultados muito lineares, sendo o mais rápido, também, o express session, porém a diferença entre o segundo mais rápido e mais lento, são, respectivamente, apenas 1.07 e 1.20 vezes mais rápido.

Tabela 17 – Duração Média em ms de execução da Rota

Rota	Express		Hapi		Koa	
	JWT	Session	JWT	Session	JWT	Session
/	783,4	916,2	1045,3	1108,8	905,9	858,4
/filme/:id	804,3	791,4	828,0	881,2	804,7	735,6
/perfil	799,8	662,9	824,8	931,0	872,7	703,8
/login	1.298,0	1155,7	1.244,7	1.457,3	1.505,7	1.439,8

Figura 15 – Gráfico Duração Média (ms)

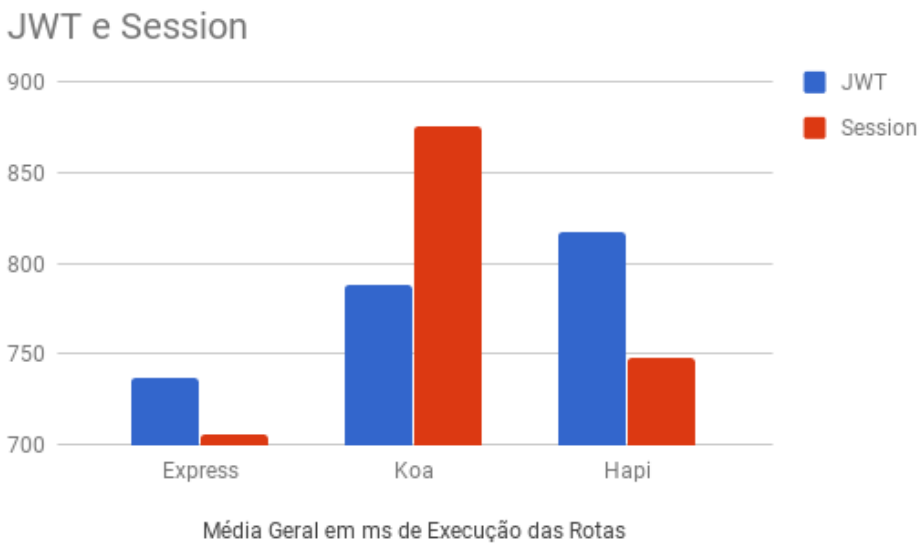


Mesmo o express session obtendo destaque entre os resultados tanto no primeiro caso de teste como no segundo caso de teste, existem dados divergentes que devem ser considerados e analisados, como as diferença entre as métricas obtidas para a rota principal. Porém, mesmo esses dados existindo, o resultado final não é alterado. Como demonstrado na Tabela 18, o mais rápido de todos para a média geral das rotas agrupados por estratégia de autenticação é o express session. Em comparação com o segundo mais rápido a diferença é quase insignificante, sendo apenas 1.04 vezes mais rápido e apenas 1.2 vezes mais rápido que o mais lento.

Tabela 18 – Média Geral em ms de Execução das Rotas

	JWT	Session
Express	737,10	705,2
Koa	788,6	875,7
Hapi	817,8	747,5

Figura 16 – Média Geral (ms)



6 CONCLUSÕES E TRABALHOS FUTUROS

Percebe-se, com base nos dados mostrados e analisados no Capítulo de Resultados, que a estratégia de autenticação *Session* foi mais eficiente que a estratégia de autenticação JSON Web Token independente do framework de desenvolvimento. *Session* obteve os melhores resultados para todos as rotas testadas no primeiro caso de teste. Além disso, para o segundo caso de testes essa estratégia obteve os melhores resultados de 3 rotas num total de 4 , sendo a rota de exceção uma rota que não utiliza estratégia de autenticação. Sendo assim, *Session* demonstrar ser a melhor opção para um sistema que necessite de eficiência na entrega de uma resposta.

Conforme mostrado na Tabela 15 para o primeiro caso de teste e na Tabela 18 para o segundo caso de teste no capítulo de Resultados, é feita uma média geral do tempo de execução das rotas agrupados por estratégia de autenticação. Sendo o Session a estratégia de autenticação mais eficiênte, o framework mais eficiente é o que obtiver a média de execução mais baixa. Sendo assim, o framework mais adequado para se utilizar a estratégia *Session* é o Express.js.

A fim de obter dados mais satisfatórios os próximos passos desse projeto deve considerar (i) teste de carga individuais para cada framework de desenvolvimento, (ii) implementação de um cluster para o MongoDB com autenticação, (iii) Teste de carga em toda a aplicação considerando a estratégia de autenticação. Pois, para (i) o resultado obtido para a rota principal demostra que existe uma diferença grande entre a performance da rota quando os parâmetros quantitativos da execução dos testes são alterados, esse comportamento não é visto nas demais rotas testadas. Para (ii) implementação do Sharding no MongoDB com autenticação não foi possível de ser feito para os ambientes de Deploy da aplicação. E (iii) pois os testes de carga realizados com o k6 demonstram que a aplicação fica instável quando o número de usuário simultâneos ultrapassa mais de mil usuários. Quando esse dado é alcançado o banco de dados gera problemas de conexão com servidor, mesmo ele estando operante.

REFERÊNCIAS

ALEXANDRE, L. C. J. **NoSQL no Suporte à Análise de Grande Volume de Dados**. 2013. Disponível em: <http://inqueritos.lead.uab.pt/OJS/index.php/RCC/article/view/61/49>.

AUTH0. **Introduction to JSON Web Token**. 2017. Disponível em: <https://jwt.io/introduction/>.

AVRAM, A. Paypal mude de java para javascript. **InfoQ**, Brazil, Dezembro 2013. Disponível em: <https://www.infoq.com/br/news/2013/12/paypal-java-javascript>. Acesso em: out. 2017.

AZZOLINI, M. G. **Desempenho de Autenticação em Aplicações Web Escaláveis no Ecossistema JavaScript Back-End**. 2017. Disponível em: <https://github.com/Mathias54/mathiasgheno_tcc>.

BASSETTE, F. **INEP Libera Site com Resultados do ENEM, mas Problemas Persistem.** 2006. Disponível em: <http://g1.globo.com/Noticias/Vestibular/0,,AA1357461-5604,00.html>.

BENAMAR ANTONIO JARA, L. L. D. E. O. N. Challenges of the internet of things: Ipv6 and network management. 2013. Disponível em: <http://ieeexplore.ieee.org/document/6975484>.

BRITO, R. W. **Bancos de Dados NoSQL x SGBDs Relacionais:Análise Comparativa**. 2010. Disponível em: <http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf>.

CAMARGO, R. G. A morte de michael jackson refletida no ciberespaÇo. **RAZÓN Y PALABRA**, Brazil, Outubro 2010. Disponível em: <http://www.razonypalabra.org.mx/N/N73/Varia73/13Graciele_V73.pdf>. Acesso em: 10 out. 2017.

CORDEIRO, A. **Morte de Michael Jackson derruba Google e outros sites**. 2006. Disponível em: <http://www.gazetadopovo.com.br/caderno-g/gente/morte-de-michael-jackson-derruba-google-e-outros-sites-bmwlqrtc6p9f5ed9cx6cr6jim>.

DAHL, R. **Introduction to Node.js**. 2009. Disponível em: <https://www.youtube.com/watch?v=ztspvPYybIY>.

DAHL, R. **node.js**. 2009. Disponível em: <http://tinyclouds.org/>.

DEBILL, E. **Module Counts**. 2017. Disponível em: <http://www.modulecounts.com/>.

EXCHANGE, S. **StackOverflow Trends**. 2017. Disponível em: <https://insights.stackoverflow.com/trends>.

FACEBOOK, i. **Privacidade Básica no Facebook**. 2017. Disponível em: <https://www.facebook.com/about/basics>.

FIELDING, R. T. **Representational State Transfer (REST)**. 2009. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>.

FOUNDATION, N. **Node.js Helps NASA Keep Astronauts Safe and Data Accessible**. 2017. Disponível em: <https://foundation.nodejs.org/wp-content/uploads/sites/50/2017/09/Node_CaseStudy_Nasa_FNL.pdf>.

FOUNDATION, N. **Walmart Strives to Be an Online Retail Leader with Node.js**. 2017. Disponível em: <https://foundation.nodejs.org/wp-content/uploads/sites/50/2017/09/Node_CaseStudy_Walmart_final-1.pdf>.

GIENOW, M. **Mikeal Rogers: Node.js Will Overtake Java Within a Year**. 2017. Disponível em: <https://thenewstack.io/open-source-profile-mikeal-rogers-node-js/>.

GITHUB. **Language Trends on GitHub**. 2015. Disponível em: <https://github.com/blog/2047-language-trends-on-github>.

GLEICK, J. **A informação: Uma história, uma teoria, uma enxurrada**. [S.l.]: Companhia Das Letras, 2013.

GRENWALD, G. **Sem Lugar Para Se Esconder**. [S.l.]: Primeira Pessoa, 2013.

GYőRöDI ROBERT GYőRöDI, G. P. A. O. C. A comparative study: Mongodb vs. mysql. 2015. Disponível em: <http://ieeexplore.ieee.org/document/7158433/>.

HAPI. **hapi.js**. 2017. Disponível em: <http://hapijs.com/>.

HARRELL, J. Node.js at paypal. **PayPal Blog**, EUA, November 2013. Disponível em: <https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/>. Acesso em: out. 2017.

KOA. **koa.js**. 2017. Disponível em: <http://koajs.com/>.

LEI YINING MA, Z. T. K. **Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js**. 2014. Disponível em: <http://ieeexplore.ieee.org/document/7023652/>.

MARROQUIM, R. M. R. M. S. C. Distribuição de dados em escala global com cassandra. 2012. Disponível em: <http://mariomarroquim.github.io/research/artigo-mariomarroquim-cassandra.pdf>.

MONGODB. **MongoDB Replica Set**. 2017. Disponível em: <https://docs.mongodb.com/manual/replication/>.

MONGODB. **MongoDB Sharding**. 2017. Disponível em: <https://docs.mongodb.com/manual/sharding/>.

NPM. **NPM Trends - express vs koa vs hapi**. 2017. Disponível em: <http://www.npmtrends.com/express-vs-koa-vs-hapi>.

NPM. **NPM Trends - koa vs hapi**. 2017. Disponível em: <http://www.npmtrends.com/koa-vs-hapi>.

PADMANABHAN, S. How we built ebay's first node.js application. **E-Bay Tech Blog**, EUA, Julho 2013. Disponível em: <http://www.ebaytechblog.com/2013/05/17/how-we-built-ebays-first-node-js-application/>. Acesso em: out. 2017.

POLITOWSKI, V. M. C. Comparação de performance entre postgresql e mongodb. 2013. Disponível em: <https://www.researchgate.net/publication/261871960_Comparacao_de_Performance_entre_PostgreSQL_e_MongoDB>.

RUPLEY, S. Node.js: An open source tool on the rise at ebay. **E-Bay News Team**, EUA, Setembro 2016. Disponível em: <http://www.ebaytechblog.com/2013/05/17/how-we-built-ebays-first-node-js-application/>. Acesso em: out. 2017.

TWITTER, I. **Privacidade no Twitter**. 2017. Disponível em: <https://help.twitter.com/pt/safety-and-security>.