DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento de uma rede social utilizando Erlang e a arquitetura Flux

Relatório submetido à Universidade Federal de Santa Catarina

como requisito para a aprovação na disciplina

DAS 5511: Projeto de Fim de Curso

Ramon Diogo Gondim Miaja Gomes

Florianópolis, Agosto de 2016

Desenvolvimento de uma rede social utilizando Erlang e a arquitetura Flux

Ramon Diogo Gondim Miaja Gomes

Esta monografia foi julgada no contexto da disciplina
DAS5511: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação

Carlos Barros Montez

_____
Assinatura do Orientador

Agradecimentos

Agradeço a meu pai e minha mãe que, mesmo passando por inúmeras dificuldades, não desistiram de mim e confiaram em minhas decisões até hoje. Agradeço também à minha companheira para vida, Carolina, por estar sempre comigo e lutar nossas batalhas de cabeça erguida. Um agradecimento especial para toda a minha turma *10.1.* Sem eles eu provavelmente não estaria me formando. São amigos para guardar debaixo de sete chaves. Agradeço a Arthur Gondim, Alex Andrade, Hugo Fagundes, Marcelo Sousa, Rodrigo Gesser e Vinícius D'all Agnol pela grande parceria durante todos esses anos de faculdade, esses guardo debaixo de oito chaves.

Resumo

O objetivo deste projeto é desenvolver uma rede social capaz de melhorar o consumo e produção de conteúdo na internet, visto que há uma clara sobrecarga de informação no dia-a-dia do usuário. No documento são discutidos os requisitos do sistema e é chegada a conclusão que a linguagem mais indicada é *Erlang*, desenvolvida pela Ericsson na década de 90. Para o desenvolvimento da camada visual da aplicação, *Vue.js,* uma biblioteca de *Javascript* que compila para três linguagens, se mostrou um ótimo candidato em conjunto com *Vuex*, uma ferramenta que utiliza as vantagens de *Vue* para aplicar a arquitetura *Flux*, desenvolvida pelo *Facebook*. Além de todo e desenvolvimento e apresentação dos componentes e modelos da rede, são apresentados resultados de pequenas ações de divulgação com dados de usuários reais.

Abstract

The goal of this project is to develop a social network that can improve the consumption and production of content on the internet, as there is a clear information overload to many people using it. In the document the system requirements are discussed and the conclusion is that Erlang – developed by Ericsson in the 90's – is the most appropriate language for its development. As for the development of the visual layer of the application, *Vue.js*, a Javascript library that compiles to three languages, was a great candidate together with Vuex, a tool that uses the advantages of Vue to apply the Flux architecture developed by Facebook. Besides the development and presentation of components and models of network, the results of small dissemination actions are shown, with real user data.

Sumário

Capítulo 1: Introdução

A última década demonstrou claramente o papel crescente da Internet na geração de um crescimento econômico global e sua importância para o comércio internacional. Economistas e tecnólogos se referem à Internet como um "possibilitador tecnológico de uso geral" – uma ferramenta que muda fundamentalmente como atividades econômicas são organizadas e que possibilita saltos de produtividade. Seu desenvolvimento permitiu a emergência de novos modelos de negócios, processos, invenções e serviços, além de aumentar a competitividade e flexibilidade na economia [1]. Segundo a Organização para Desenvolvimento e Cooperação Econômica, o impacto da Internet na produtividade é maior que o impacto de qualquer outro "possibilitador tecnológico" na história, incluindo a eletricidade e o motor a combustão [2].

Durante a fase de desenvolvimento e maturação da Internet, algumas empresas se destacaram ao dominar os meios de informação. *Google*, *Facebook* e *Yahoo* são exemplos de negócios que passaram a definir como o conteúdo na Internet é consumido e produzido.

Empresas e pessoas passaram a adotar uma nova estratégia para disseminar suas ideias e opiniões. O *marketing* digital se tornou um novo modelo de negócios, possibilitando um meio completamente inovador de ganhar dinheiro. Somando isso ao desenvolvimento de novas redes sociais, tem-se um sistema completamente autossuficiente onde pessoas compartilham informações, compram e vendem produtos, trabalham e consomem conteúdo. O crescimento excessivo desse sistema acabou por gerar uma grande complicação: muita informação para pouco tempo de consumo.

Tendo esse problema como motivação, a proposta do projeto deste documento é criar uma **ferramenta que facilite ao usuário o acesso rápido e prático a conteúdos de seu interesse**, independentemente de onde foi criado ou divulgado. Para isso foi desenvolvida uma rede social denominada Instablah, a qual espera-se que seja a ferramenta que faltava nesse sistema de consumo e produção de conteúdo.

Neste documento estão descritas todas as etapas do processo de criação da rede, da idealização até o estado atual (em produção). Os capítulos são separados em:

- Formalização do problema
- Formalização da solução
- Tecnologias e metodologias
- Descrição conceitual técnica
- Resultados do projeto
- Conclusões e perspectivas

Capítulo 2: Formalização do problema

Conforme se torna mais fácil o acesso a ferramentas para produção, divulgação e consumo de conteúdo, a quantidade de publicações cresce cada vez mais. Artigos em *blogs*, vídeos *online*, imagens e sistemas de mensagens substituíram jornais e programas de televisão. A mídia digital invadiu a vida das pessoas, tendo a diversidade como seu carro-chefe e proporcionando um enriquecimento da interatividade do usuário com a informação.

2.1: A primeira rede social

Considerada a primeira rede social da internet, a *SixDegrees* foi fundada por Andrew Weinreich em maio de 1996. No ano seguinte, em seu lançamento, as pessoas puderam criar seus perfis, adicionar amizades e, em 1998, navegar por sua lista de amigos. Cada uma desses atributos já existia previamente. A opção de ter perfil já era presente na maioria dos sites de relacionamentos amorosos e diversos *fórums online*. AIM e ICQ já aceitavam lista de amizades, embora essa lista não fosse visível ao público. A rede Classmates.com já permitia aos usuários se afiliarem às suas antigas escolas ou universidades e navegar para encontrar colegas de classe, porém a rede não tinha a opção de fazer um perfil público nem ter uma lista de amizades (pelo menos na época) [3].

Figura 1 - Logo da SixDegrees

A *SixDegrees* foi a primeira rede que reuniu todas essas características em um só produto. A empresa promovia a si mesmo como uma ferramenta para ajudar as pessoas a se conectarem. Mesmo atraindo milhões de usuários, não conseguiu se sustentar e acabou fechando em 2000. Seus fundadores acreditam que a rede estava muito à frente de seu tempo. Enquanto as pessoas estavam começando a usar a internet, a maioria não possuía amigos com o mesmo acesso e acabavam por não ter muito o que fazer no site.

2.2: A nova mídia social

Após o aparecimento dos *blogs*, a mídia social começou a expandir. Sites como *Friendster*, *MySpace* e *LinkedIn* ganharam proeminência e ferramentas como *Photobucket* e *Flickr* facilitaram o compartilhamento de imagens entre pessoas. Em 2005 é fundado o *YouTube*, criando uma nova maneira de comunicação por vídeos e facilitando o compartilhamento de conteúdo entre as pessoas. Em 2006, nascem os gigantes *Facebook* e *Twitter* que hoje possuem o posto de maiores e mais bem-sucedidas redes sociais do mundo.

2.2.1: O Gigante Facebook

Lançada em 4 de fevereiro de 2004, a rede social *Facebook* é considerada a maior rede social do mundo [4]. Fundada por Mark Zuckerberg e seus colegas de quarto, seu uso foi limitado aos estudantes de Harvard, sendo depois expandido para o público.

Figura 2 - Logo do Facebook

Dentre seus números extravagantes, estão:

- 1.65 bilhões de usuários ativos por mês em março de 2016 (com um aumento de 15% por ano).
- 1.09 bilhões de usuários ativos por dia em março de 2016.
- 5 contas são criadas a cada segundo.
- 300 milhões de fotos são postadas por dia.
- 4.75 bilhões de postagens são compartilhadas por dia (maio de 2013).
- 16 milhões de páginas de negócios locais foram criadas até maio de 2013, com um crescimento de 100% em relação a junho do ano anterior. [5]

Com números tão expressivos, é possível imaginar o tamanho da dificuldade em filtrar o conteúdo. Algoritmos de recomendação e outras técnicas de inteligência artificial foram postos à prova para esse desafio e mesmo assim o problema se mostra mais complicado a cada dia que passa.

2.3: Sobrecarga de conteúdo

Com pouca (ou nenhuma) informação, um indivíduo possui poucos dados para processar e consequentemente toma decisões ruins. Conforme a quantidade de informações aumenta, intensifica-se o processamento e a qualidade da tomada de decisões melhora. No entanto, após certo ponto o tomador de decisões recebe mais informação do que consegue processar. Ocorre então uma sobrecarga de informação. O que for recebido a partir desse ponto não será processado e poderá levar à confusão, prejudicando a habilidade de definir prioridades e lembrar de informações passadas [6].

O conceito de sobrecarga de informação exemplifica um grande problema enfrentado por pessoas e empresas hoje em dia. Há muita informação, mais do que se deseja e mais do que se consegue processar. Um usuário, ao entrar em um site, é normalmente bombardeado de conteúdo que não lhe interessa. Algo que poderia se tornar uma oportunidade de entrar em contato com algo interessante, acaba virando mais uma página fechada no navegador. Para empresas ainda é pior, já que

pagam caro para divulgar seus produtos e ideais pela Internet. No fim, acaba gastando mais para atingir pouco.

Com 3.4 bilhões de pessoas usando a Internet [7], a produção de conteúdo cresce cada vez mais. Seja um comentário, uma postagem ou uma mensagem, qualquer pedaço de informação contribui para inflar a bolha do consumo digital. Quem gera conteúdo quer ser visto, consumido e compartilhado e quanto melhor for o retorno de seus consumidores, maior é o incentivo a gerar mais informação. Resta aos serviços de compartilhamento e redes sociais tentar filtrar, sem muito sucesso, essa massiva quantidade de conteúdo para enfim direcioná-la a seu público alvo.

Capítulo 3: Formalização da solução

Com os problemas discutidos, a motivação do projeto toma forma: filtrar o melhor conteúdo sobre determinado assunto para o usuário. Esse capítulo foca em explicar as etapas para a solução do problema de acordo com possíveis filosofias e funcionalidades do projeto.

3.1: Categorização de conteúdo

Há muitos serviços e sites na internet que separam o conteúdo que criam em categorias, facilitando o acesso à assuntos de interesse. É um método simples: quando um artigo é criado, o escritor define qual sua categoria. O problema surge ao tentar classificar conteúdos de outro site, para esse caso, o elevado fluxo de informação torna esse método de categorização algo fora de cogitação.

Pensando na solução para esse problema, deve ser desenvolvido um método de classificação público, tendo o usuário como agente classificador. Assim, abstrai-se essa responsabilidade, fazendo com que o próprio usuário defina a classe de um conteúdo. Para fortalecer esse sistema, é interessante possibilitar a outros usuários decidir se tal classificação faz sentido, permitindo assim um sistema de votação para as classificações.

Figura 3 - Classificação por votação

Ao passar a responsabilidade de classificação ao usuário, surge o problema de ficar à mercê de opiniões muito enviesadas ou atos de má fé. Tais problemas podem ser resolvidos com um sistema de moderação, onde há sempre uma pessoa (ou um grupo de pessoas) responsável por verificar se as classificações estão de acordo com o conteúdo.

3.2: Qualificação do conteúdo

Após a etapa de classificação, é preciso saber se um conteúdo específico tem qualidade. O *Facebook* é conhecido por ter resolvido esse problema. Seu botão de “curtir” é um mecanismo fundamental pata medir a popularidade de certo conteúdo na Internet [8]. Essa funcionalidade acabou se popularizando pela internet e seu sucesso (cerca de 3 bilhões de cliques por dia [9]) mostrou que a opinião pública é um bom meio de avaliar um pedaço de informação.

Figura 4 - Botão de "curtir" do Facebook

Considerando o peso da opinião pública no julgamento da qualidade de um conteúdo, foi decidido que grande parte do sistema de qualificação deve ser guiado pelos usuários que têm interesse pela classe daquele conteúdo. Se uma foto ou artigo sobre futebol deve ser julgado, que seja julgado por pessoas que gostem de futebol.

3.3: Interatividade

Garantida a criação e classificação do conteúdo, resta definir como o usuário interage com ele. Dado o curto espaço de tempo entre o começo do projeto e sua apresentação, foi decidido incluir apenas o atributo de poder comentar determinado conteúdo, seguindo os mesmos moldes observados em outras redes sociais, como *Facebook* e *Instagram*.

3.4: Estrutura do site

Resumindo as funcionalidades esperadas para uma primeira versão, tem-se uma lista do que o Instablah deve proporcionar ao usuário:

- Criar e apagar conteúdo.
- Compartilhar conteúdo de outras fontes.
- Classificar conteúdo (futebol, arte, música).
- Qualificar conteúdo.
- Comentar sobre um conteúdo.
- Compartilhar conteúdo.

Com base nessa especificação, é possível desenvolver e estruturar atributos que tornem possível a interação do usuário com o site. A seguir, são discutidos esses atributos.

3.4.1: Postagem

Uma postagem é o elemento principal de visualização do conteúdo no Instablah. Uma postagem pode conter texto, imagem, vídeo e links para outros sites, além de poder ser comentada, classificada e qualificada.

3.4.2: Canal

Um canal é uma página que agrega todas as postagens relacionadas a seu nome. Sendo a única forma de acesso ao conteúdo, o site é composto por vários canais que se conectam, possibilitando ao usuário navegar pela rede. Com isso, espera-se que postagens sobre futebol estejam dentro do canal “#futebol” e que canais relacionados estejam listados em alguma parte da página. Além disso, o usuário pode criar quantos canais quiser. O símbolo “#” é usado para identificar o nome de um canal.

3.4.3: Persona

A persona é o atributo que define o usuário. Uma persona possui um nome e uma imagem como elementos principais, além de ser o único atributo a tomar ações no site (votar, comentar, criar canal, etc.). Por não conter nenhuma espécie de algoritmo de qualificação ou classificação de conteúdo, o site confia apenas às personas essa responsabilidade.

3.4.4: Voto

O voto é o atributo usado para definir a qualidade de uma postagem. Há dois tipos de votos: o voto positivo e o negativo. A estrutura é simples, quanto maior a diferença entre votos positivos e negativos, maior a qualidade.

3.4.5: Comentário

Além de votar, o Instablah deve oferecer ao usuário a opção de comentar em uma postagem. Essa interação enriquece a experiência de consumo de conteúdo ao torná-la mais social.

3.4.6: Estruturas futuras

A definição da estrutura da primeira versão é importante, porém é imprescindível considerar características futuras da rede para auxiliar na escolha das tecnologias usadas em seu desenvolvimento.

Dentre atributos futuros estão:

- Comentários em tempo real (sendo tempo real definido como a prática de um usuário receber informação sem precisar recarregar a página em seu navegador)
- Lista de postagens de outros canais, recomendadas por outros usuários
- Ferramenta de criação de postagens mais customizadas
- Criação de postagens em tempo real para uma audiência ao vivo
- Sistema de troca de mensagens
- Sistema de notificações
- Sistema de e-mail
- Versão móvel
- Suporte para propagandas personalizadas
- Sistema de moderação
- Sistema de recompensas

Baseado nas especificações apresentadas, o próximo capítulo detalha o processo de escolha das tecnologias a serem usadas no desenvolvimento da rede.

Capítulo 4: Tecnologias e metodologias

Como todo projeto de desenvolvimento de site, é preciso definir linguagens de programação, bibliotecas, banco de dados e frameworks a serem usados. Sabendo que o fluxo de dados de uma rede social é grande e que muitas requisições ocorrem simultaneamente, as tecnologias escolhidas devem possuir características que facilitem o desenvolvimento nesse sentido. Além disso, é imprescindível que, seguindo metodologias modernas, o site seja reativo, ou seja, que toda ação realizada pelo usuário desencadeie uma série de reações na página, contribuindo para uma usabilidade mais fluida e agradável.

Com base nas especificações, é possível criar uma lista de requisitos técnicos:

- O site deve aguentar um grande número de requisições simultâneas e concorrentes
- O sistema deve ser de fácil distribuição entre uma rede de computadores
- O sistema deve escalar de acordo com o número de máquinas na rede
- O sistema deve permitir ser modificado, configurado e melhorado sem que tenha que ser reiniciado
- O sistema não pode cair

Considerando as requisições, a experiência dos programadores e ferramentas atuais, Erlang foi escolhida como linguagem base para o desenvolvimento da estrutura da aplicação, servidor e comunicação com banco de dados. Para desenvolver a estrutura da página no navegador, bem como o comportamento e o design dos elementos visíveis ao usuário, foram utilizados HTML, CSS, Javascript e sua biblioteca *Vue.js*.

4.1: Erlang

4.1.1: Por que usar Erlang?

Se o sistema desejado for um sistema em tempo real de alto nível, concorrente, robusto, leve, que escale conforme necessidade, que faça uso completo dos processadores e seja integrável a outras linguagens, então Erlang é, provavelmente, a melhor escolha [10].

Figura 5 - Logo de Erlang

Empresas que usam Erlang:

- Amazon usa Erlang para implementar o SimpleDB, provendo serviços de banco de dados ao *Amazon Elastic Compute Cloud* (EC2)
- Yahoo! o usa em seu serviço de *bookmarking*, o Delicious, possuindo mais de 5 milhões de usuários e 150 milhões de *URLs*
- O Facebook usa Erlang em seu serviço de *chat*, cuidando de mais de 100 milhões de usuários ativos.
- T-Mobile usa Erlang em seu sistema de SMS
- Motorola e Ericsson usam Erlang para desenvolver alguns de seus produtos.

A Universidade de Uppsala, na Suécia, liderou por muitos anos as pesquisas na linguagem através do HiPE (*High Performance Erlang Project*), mas muitas outras universidades pelo mundo já estão com projetos de pesquisas bem avançados na área. Entre elas estão a Universidade de Kent (Reino Unido), Eötvös

Loránd University (Hungria), Universidade Politécnica de Madrid (Espanha), Chalmers University e IT University (as duas na Suécia).

4.1.2: História de Erlang

Em meados da década de 1980, o Laboratório de Ciência da Computação da Ericsson recebeu a tarefa de investigar linguagens de programação adequadas para o desenvolvimento da próxima geração de produtos de telecomunicação. Joe Armstrong, Robert Virding, e Mike Williams, sob a supervisão de Bjarne Däcker, passaram dois anos prototipando aplicações com todas as linguagens de programação disponíveis da época. A conclusão foi que, embora muitas delas tivessem características interessantes e relevantes, nenhuma cobria por completo as reais necessidades da época. Como resultado, decidiram inventar a sua própria linguagem.

Erlang foi influenciada por linguagens funcionais como ML e Miranda, linguagens concorrentes, como ADA, Modula e Chill, assim como a linguagem de programação lógica Prolog. As propriedades de atualização de software de Smalltalk também desempenharam seu papel, assim como as linguagens proprietárias da Ericsson: EriPascal e PLEX.

Com uma máquina virtual baseada em Prolog, o laboratório prototipou, por quatro anos, aplicações de telecomunicação em uma linguagem que, por tentativa e erro, se tornou o Erlang que é conhecido hoje. Em 1991, Mike Williams escreveu a primeira máquina virtual em C e, um ano depois, o primeiro projeto comercial foi iniciado. Em 1994, o lançamento do produto permitiu aos desenvolvedores da linguagem um conhecimento maior sobre quais melhorias deveriam ser feitas e que atributos deveriam ser incorporados em uma nova versão.

Após o sucesso do primeiro projeto comercial e a maturação da linguagem, Erlang começou a ser usado em grandes projetos, como a banda larga da Ericsson, GPRS e ATM (também da Ericsson). Paralelamente a esses projetos, a framework OTP (*Open Telecon Platform*) foi desenvolvida e lançada em 1996. A OTP provê um framework para estruturar sistemas em Erlang, oferecendo robustez e tolerância a falhas (e ainda muitas bibliotecas e ferramentas) [11].

4.1.3: A importância de sua história

A trajetória da criação de Erlang é muito importante para entender sua filosofia. Ao contrário de muitas linguagens, Erlang foi desenvolvida para resolver requerimentos do mercado, sendo que sistemas distribuídos, tolerantes a falhas, massivamente concorrentes e em tempo real estavam começando a se tornar necessários. O fato de serviços *web*, bancos, telefonia digital, sistemas de mensagens e integração em empresas compartilharem os mesmos requerimentos de serviços de telecomunicações explica o porquê de Erlang estar ganhando tração nos últimos anos.

A Ericsson tomou a decisão de tornar a sua linguagem em um projeto de software livre em 1998 (o que foi feito sem anúncios midiáticos e sem gastar ou ganhar nada). O crescimento de Erlang se deve a uma crescente comunidade, resultada da combinação de projetos de pesquisa, projetos comerciais e softwares de código aberto, assim como sua divulgação por blogs, *marketing* viral e livros, todos guiados pela necessidade de resolver difíceis problemas na área para qual Erlang foi criada.

4.1.4: Processos concorrentes

Concorrência em Erlang é fundamental para seu sucesso. Ao invés de *threads* que compartilham memória, cada processo em Erlang é executado em sua própria memória e é dono de sua própria pilha de chamada. Processos não podem se interferir inadvertidamente, como é fácil de ser feito em modelos que usam threads, levando a *deadlocks* e outros problemas.

A comunicação entre processos é feita via passagem de mensagens, onde a mensagem pode ser qualquer tipo de dado em Erlang. Esse mecanismo é assíncrono, então quando uma mensagem é enviada, o processo pode continuar seu processamento. As mensagens são recebidas e armazenadas seletivamente, o que torna não necessário seu processamento na ordem em que chegaram. Isso aumenta a robustez do modelo de concorrência, principalmente em sistemas onde processos são distribuídos em vários computadores e a ordem de chegada das mensagens depende muito da performance da rede.

4.1.5: Escalável, seguro e eficiente

O sistema de concorrência em Erlang é rápido e escalável. Seus processos são tão leves quanto conseguem ser e sua MV (máquina virtual) não cria um *thread* no SO (sistema operacional) para cada processo criado. Todos são criados, agendados e manejados na própria MV, independente do SO. Como resultado, o tempo de criação de um processo é da ordem de microssegundos e independente do número de processos concorrentes existentes. Em C# e Java, cada processo criado gera um novo thread no SO, o que leva a muitos problemas de performance e processamento.

O tempo de passagem de mensagens também é da ordem de microssegundos e independe do número de processos existentes. A troca pode ser resumida em copiar os dados da memória de um processo para o espaço de memória do outro, tudo na mesma MV. Esse método é diferente do adotado em Java e C#, que trabalham com memória compartilhada, semáforos e threads do SO. Aqui, estudos de caso mostram que Erlang supera essas linguagens em termos de segurança e velocidade de processamento [12].

4.1.6: Robustez

A busca por programas robustos acaba tomando muito tempo de seus programadores. Quanto maior o projeto, mais defensivo deve ser o programador em relação a possíveis *bugs* no sistema. Pensando nisso, Erlang apresenta uma série de mecanismos e ferramentas que facilitam esse trabalho. Bibliotecas simples e robustas foram criadas para tratar erros e garantir que o sistema não desligue. Essas bibliotecas são conhecidas como OTP *middleware*. Ao programar visando apenas a solução do problema, deixando de lado métodos defensivos, não apenas os programas são menores e mais fáceis de entender, como geralmente contém menos *bugs*.

Processos em Erlang podem ser conectados de maneira a garantir que se um quebre, o outro seja informado e possa tomar ação (tratando o erro ou terminando a si mesmo). O OTP contém uma série de comportamentos genéricos, como servidores, máquinas de estado finito e tratadores de eventos que se comportam como processos "trabalhadores", sendo supervisionados por processos

"supervisores", cuja única responsabilidade é monitorar e terminar outros processos. Essa supervisão e conexão entre processos garante que erros possam acontecer sem danificar o comportamento do sistema em geral.

4.1.7: Integração com demais linguagens

Erlang possui avançadas bibliotecas especializadas em comunicar a aplicação com pedaços de código em outras linguagens. Isso garante que as partes negativas da linguagem possam ser superadas e que o programa tenha total liberdade de se comunicar com outras aplicações.

4.1.8: Erlang e Multicore

A mudança para sistemas que utilizem a total capacidade de todos os núcleos de uma máquina é inevitável no mundo. Erlang e seu modelo concorrente – processos separados sem compartilhar memória comunicando-se por troca de mensagens – são naturalmente aceitos por múltiplos processadores de uma maneira muito transparente para o programador, possibilitando o desenvolvimento de aplicações sem um eventual redesign voltado à multicore.

O suporte para multiprocessamento simétrico (SMP) em Erlang foi primeiramente desenvolvido no final da dédaca de 90 e é agora uma parte integral do lançamento padrão. O etos da equipe de desenvolvimento era fazer o SMP funcionar, medir sua performance, achar gargalos e otimizar. Desde a primeira versão, essa tem sido sua abordagem. Em versões recentes, o modelo da máquina virtual evoluiu de uma única fila de execução – provavelmente com processos rodando em diferentes processadores – para uma fila de execução por processador, garantindo que a fila não seja mais um gargalo ao sistema, como ilustrado na *figura 6*.

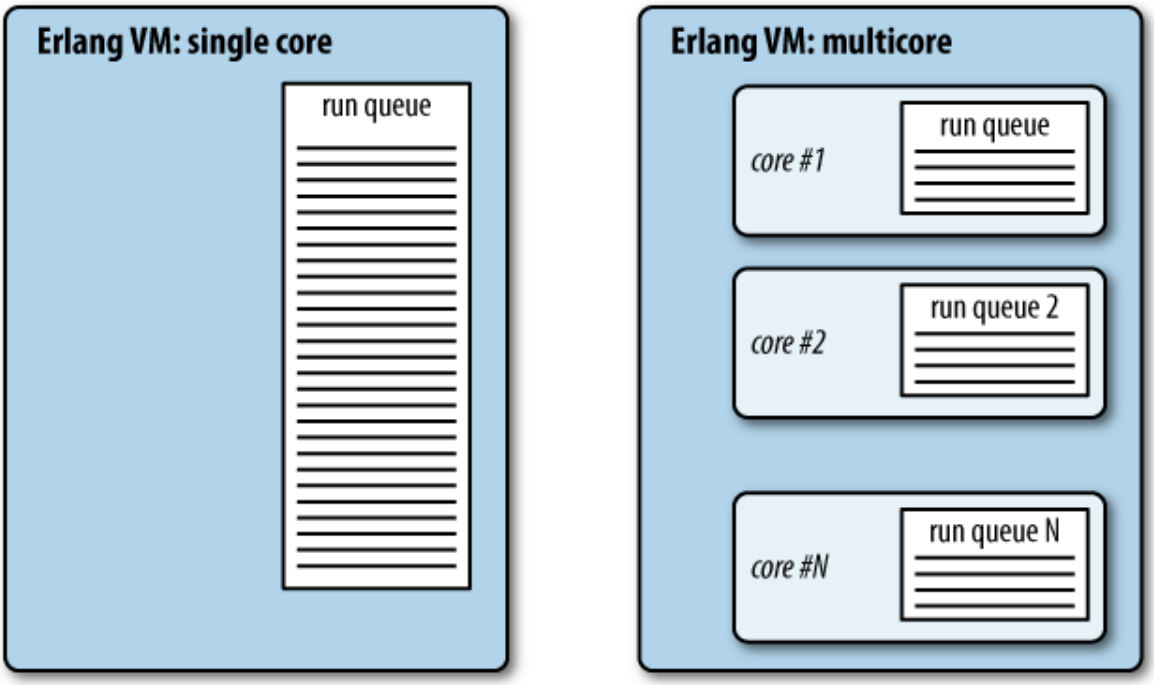

Figura 6 - Filas de execução com multiprocessadores

4.1.9: Desvantagens de Erlang

Evan Miller, conhecido programador Erlang e criador do framework Chicago Boss, escreveu em seu blog: "...então chegamos à má notícias: os benefícios de Erlang só são aproveitados após anos de uso. Certamente não é uma linguagem para iniciantes. A sintaxe é estranha para qualquer programador vindo de outra linguagem imperativa. Programação funcional é difícil e Erlang não facilita esse cenário. As ferramentas gráficas são primitivas e não nenhum tipo de curso introdutório didático como Java possui. Ler qualquer programa não trivial requer um bom conhecimento de recursão. Erlang também apresenta um número baixo de bibliotecas se comparado com outras linguagens; em minha experiência, para qualquer tarefa, há uma ou nenhuma biblioteca disponível para o trabalho. Posso ser o único a pensar assim, mas na verdade eu gosto do fato de a linguagem possuir poucas bibliotecas disponíveis. Se eu preciso de algo feito, tenho a desculpa de fazer por conta própria e, com isso, acabo aprendendo mais e descobrindo coisas que não sabia antes. Parece estúpido, mas é verdade. Me sinto produtivo pois estou fazendo algo que ninguém jamais fez" [13].

4.1.10: Chicago Boss

Figura 7 - Logo do Chicago Boss

Erlang tem a reputação de ser uma tecnologia inacessível, difícil de dominar e que é usado apenas por uma pequena elite que demorou anos para aprender sua complicada sintaxe. A verdade é que por se tratar de uma linguagem funcional, a maioria das pessoas, acostumadas com linguagens imperativas, acaba desistindo antes de começar. Ao adotar convenções familiares a *Ruby on Rails*, a framework *Chicago Boss* tenta transformar Erlang em uma ferramenta mais acessível a esse público [14].

4.1.11: Erlang e o Instablah

Tratando-se de uma rede social, o Instablah deve ser capaz de sustentar um elevado tráfego de informações, já que possibilita troca de mensagens entre usuários em tempo real. Fazendo uma breve análise de Erlang e prevendo uma elevada carga de dados no site, a linguagem se mostrou como a mais indicada para estruturar a aplicação.

4.2: Vue.js

Vue.js é uma biblioteca em javascript utilizada no desenvolvimento de interfaces web. Ao utilizar o padrão *Model View ViewModel*, seu objetivo é prover os benefícios de uma estrutura de dados reativa e componentizada com uma API (*Application Programming Interface*) simples. A biblioteca não é considerada um *framework*, visto que foca apenas na camada visual da aplicação. Mesmo assim, sua alta capacidade de integração, leveza, simplicidade, suporte externo e

crescente comunidade transformam *Vue* em uma ferramenta completa para o desenvolvimento de aplicações *web* [15].

Figura 8 - Logo do Vue.js

A principal proposta de *Vue* é transformar a camada visual da página em um sistema orientado a dados. Normalmente usa-se um paradigma imperativo para controlar elementos da página, o que torna o código repetitivo, difícil e propenso a erros. *Vue* recorre à reatividade para montar uma estrutura em que os elementos da página estão diretamente conectados a uma camada de dados que controla todo seu comportamento. Isso permite que o programador se concentre na lógica da aplicação ao invés de ter que se preocupar em modificar determinado elemento.

Outro conceito importante em *Vue* é seu sistema de componentes, sendo uma abstração que permite desenvolver grandes aplicações compostas de pequenos componentes reusáveis. Cada componente cuida apenas de seu próprio escopo, ou seja, não altera nenhum dado externo a ele. A vantagem desse conceito é que quase todo sistema pode ser abstraído em uma árvore de componentes, fazendo com que *Vue* seja uma ótima ferramenta para a realização de testes de unidade.

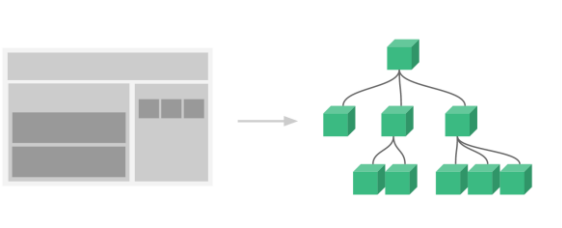

Figura 9 – Aplicação abstraída em uma árvore de dados

4.2.1: Vuex

Ao se usar *Vue*, a tendência é guardar o estado do componente dentro dele mesmo, o que garante que nenhum outro componente possa modificar seus dados e diminui as chances de erros inesperados. Nesse padrão, cada componente é responsável por um pedaço do estado geral da aplicação, o que facilita o tratamento de erros ao isolar certas partes do projeto, garantindo uma fácil identificação do problema e uma rápida solução [16].

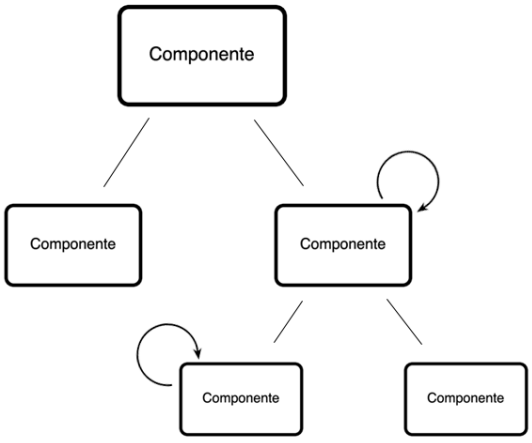

Figura 10 - Cada componente modifica a si mesmo

Para aplicações maiores – como é o caso do Instablah – não é raro que um componente tenha de ser modificado em razão de uma ação tomada em outro lugar da página. Nesse caso, é comum observar a prática de modificar dados na ocorrência de determinados eventos. Para isso um componente deve sempre esperar por algum sinal enviado para a aplicação para então modificar alguma variável. Apesar de ser uma boa saída para uma comunicação entre componentes, essa solução acaba por aumentar a complexidade da aplicação. Se houver muitos

eventos ou se a estrutura dessa comunicação for muito grande, perde-se a capacidade de localizar a origem de cada ação, o que confunde o programador e atrasa uma eventual correção de *bugs*.

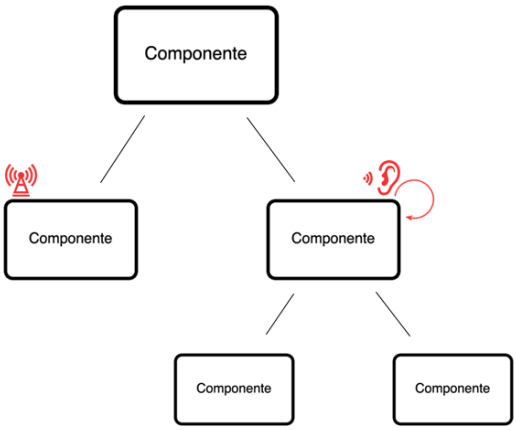

Figura 11 - Componente dispara evento enquanto outro escuta e realiza uma ação

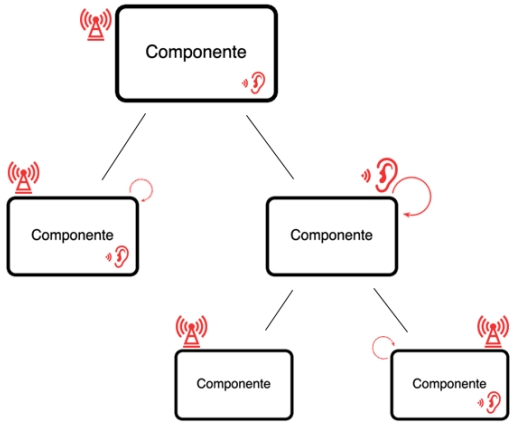

Figura 12 - Estrutura baseada em eventos tende a ficar complexa

Para resolver esse problema, foi criada uma biblioteca para *Vue*, chamada *Vuex,* inspirada na arquitetura Flux, desenvolvida por engenheiros do *Facebook*. *Vuex* provê à aplicação uma arquitetura para um gerenciamento de estado centralizado, possuindo conceitos mais simples que *Flux* e uma implementação feita especificamente para utilizar as vantagens do sistema de reatividade de *Vue*. Seu funcionamento é melhor explicado no item *4.4.1*.

4.3: PostgreSQL

PostgreSQL é um sistema de gerenciamento de banco de dados relacional de código aberto que enfatiza extensibilidade. Normalmente usado em projetos de pesquisa, o *PostgreSQL* é encontrado também em diversos produtos comerciais. Sua capacidade de armazenar e modificar estruturas em JSON acabou sendo uma das principais razões de sua escolha para o projeto [17].

4.4: Arquitetura Flux

Figura 13 - Logo da arquitetura Flux

Flux é a arquitetura de gerenciamento de dados que o *Facebook* usa para desenvolver aplicações *web*. Para introduzir *Flux* de uma maneira mais simples, é necessário entender quais problemas esse padrão resolve e quais problemas o *Facebook* enfrentou que serviram de motivação para seu desenvolvimento.

No *Facebook*, quando o usuário recebe uma mensagem de outra pessoa, é incrementado, na página, o número de mensagens não visualizadas. Ao clicar no ícone que representa essa informação, é possível ver a mensagem recebida e interagir com ela. É um sistema simples que espera por um dado e, ao receber esse dado, incrementa um número. Com o aumento da complexidade da aplicação e do número de usuários, esse sistema começou a apresentar falhas. Em muitos casos, a mensagem chegava, o contador era incrementado, mas ao clicar no ícone, nada aparecia, como se o usuário não houvesse recebido nenhuma mensagem.

Ao resolver esse problema, a equipe do *Facebook* percebeu que, não muito tempo depois, outro bug semelhante acontecia. O que era apenas um pequeno erro no sistema, tinha se tornado um ciclo constante de *bugs*, algo deveria ser feito para contornar essa situação. Os engenheiros não queriam ter de consertar o mesmo

problema várias vezes, então decidiram atacar sua raiz: o controle do estado da aplicação no navegador. Buscavam então fazer um sistema restrito onde esse tipo de erro era previsível. Por ser um código imperativo, cada ação do usuário tinha de passar por uma série de condições que determinavam uma mutação do estado. Algumas ações, inclusive, desencadeavam uma série de alterações nos modelos que produzia um emaranhado de dados sendo transformados sem nenhuma estrutura para controlar esse sistema. Acrescentando que todas essas ações poderiam estar acontecendo assincronamente, é possível perceber que esse sistema estava fadado a falhas. O *Facebook* decidiu, então, criar uma arquitetura diferente, onde os dados fluem apenas em uma direção. Cada ação do usuário deve ser previsível e deve ser passada a um sistema expedidor que informa essas ações a um controlador geral, responsável por realizar mutações no estado da aplicação e informar à página essa mudança de estado. Eles chamaram essa arquitetura de *Flux* [18].

No Instablah, essa arquitetura é usada em conjunto com a biblioteca Vuex, que acaba utilizando as propriedades reativas de *Vue* para criar uma ferramenta poderosa e de fácil compreensão.

Capítulo 5: Descrição conceitual técnica

Com as tecnologias e metodologias definidas, o desenvolvimento seguiu durante 4 meses. Este capítulo aborda a descrição técnica de cada elemento desenvolvido no projeto.

Todo desenvolvimento de site é dividido em duas partes: o *back-end* e o *front-end*. O *back-end* é a parte responsável por estruturar os modelos, controladores e rotas da aplicação. Já o *front-end* tem a responsabilidade de cuidar da camada visual e das interações do usuário com a página. Este capítulo detalha o desenvolvimento de cada elemento da rede social Instablah.

5.1: Postagem

A postagem é o principal meio de visualização de conteúdo, podendo ser composta por texto, imagem, vídeo ou por uma prévia de algum site externo. Essas 4 características determinam o tipo da postagem, sendo que para cada tipo existe uma tabela diferente no banco de dados. A seguir são apresentados o código de criação da tabela principal e as tabelas de todos os tipos de postagens.

5.1.1: Back-end

Criação da tabela posts

```
CREATE TABLE posts(
	id serial PRIMARY KEY,
	type text NOT NULL,
	slug text UNIQUE,
	persona_id integer NOT NULL REFERENCES personas
		ON UPDATE CASCADE ON DELETE CASCADE
	);
```

Tabela *Posts*

Referente à estrutura principal de uma postagem

Atributo	Tipo
id	serial
type	text
slug	text
persona_id	integer

Tabela 1 - Estrutura principal de uma postagem

Tabela *PostTexts*

Referente à estrutura de uma postagem textual

Atributo	Tipo
id	serial
title	text
text	text
text_html	text
post_id	integer

Tabela 2 - Estrutura de uma postagem textual

Tabela *PostImages*

Referente à estrutura de uma postagem com imagem

Atributo	Tipo
id	serial
title	text
text	text
text_html	text
post_id	integer
img	text

Tabela 3 - Estrutura de uma postagem com imagem

Tabela *PostLinks*/*PostVideos*

Referente à estrutura de uma postagem contendo uma prévia de um site externo ou um vídeo

Atributo	Tipo
id	serial
title	text
text	text
text_html	text
post_id	integer
source_title	text
source_host	text
source_img	text
url	text

Tabela 4 - Estrutura de uma postagem de prévia ou vídeo

5.1.2: Front-end

A postagem foi feita seguindo moldes de diferentes redes sociais, mas sempre tentando desenvolver uma identidade própria. A estrutura visual da postagem é apresentada na *figura 14* e seus pontos discutidos adiante.

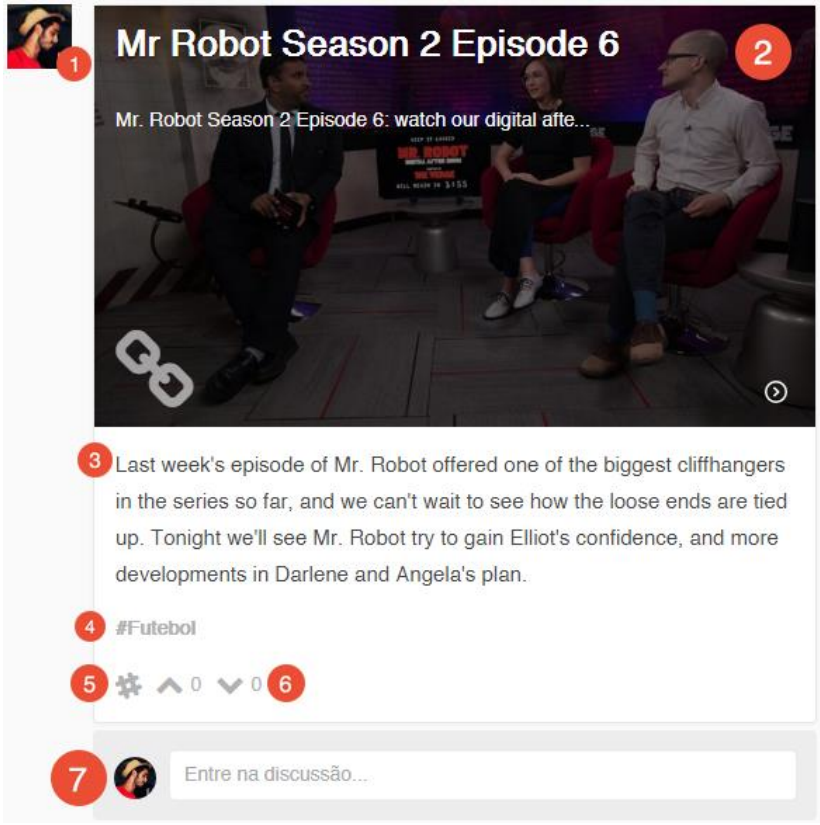

Figura 14 – Design de uma postagem

1. Imagem do usuário que postou o conteúdo.
2. Prévia de uma página externa, composta por um título, um subtítulo e um ícone indicando o tipo da postagem (nesse caso uma corrente, indicando se tratar de um *link*). Essa parte também pode conter um vídeo ou uma imagem.
3. Texto da postagem.
4. Classes da postagem. Nesse caso, ela está presente no canal *#Futebol.*
5. Botão para classificar conteúdo.
6. Botões para qualificar a postagem, podendo ser positivo ou negativo.

7. Área para realizar um comentário.

Os quatro tipos de postagem possuem leves variações de design. Como mostrado nas *tabelas 2, 3 e 4*, cada um deles apresenta variáveis semelhantes, porém são desenhados de maneira diferente na camada de visualização.

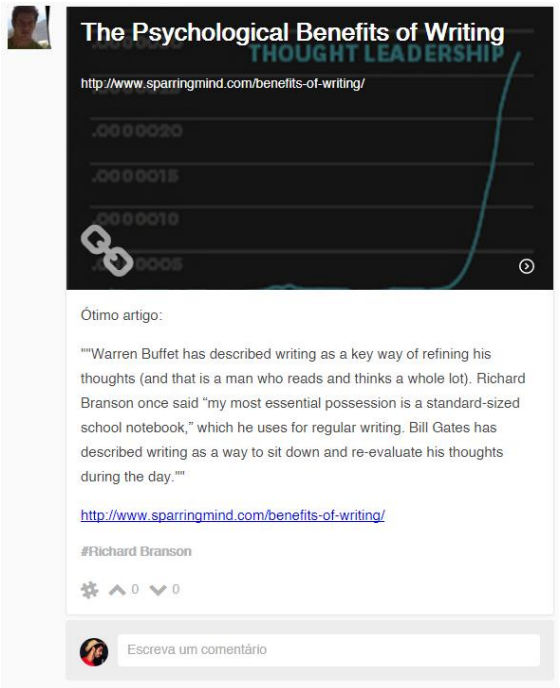

Figura 15 – Postagem em formato de link

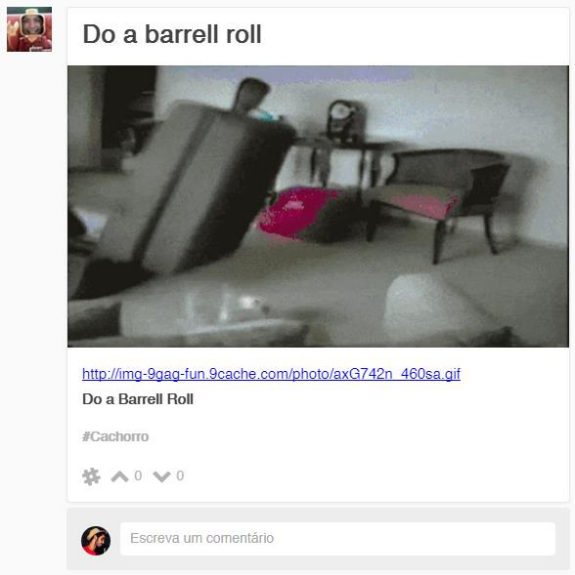

Figura 16 – Postagem em formato de imagem

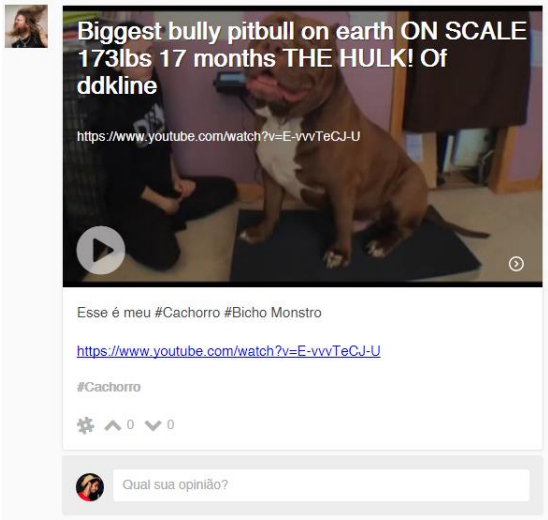

Figura 17 – Postagem em formato de vídeo

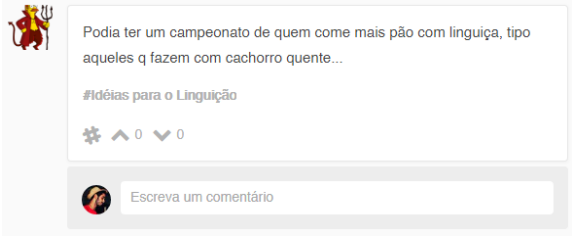

Figura 18 – Postagem em formato de texto

É importante ressaltar que uma postagem é um componente da aplicação. Abstraindo seu comportamento, tem-se sua “árvore de componentes”:

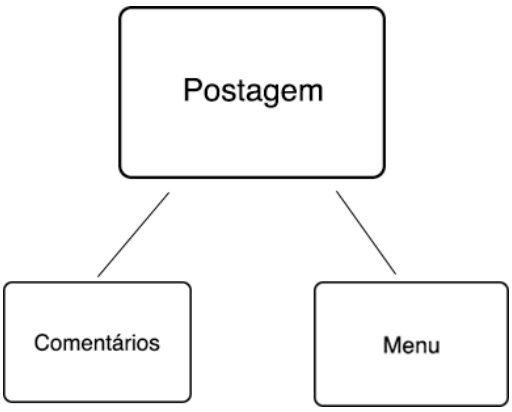

Figura 19 - Árvore de componentes de uma postagem

A divisão dessa estrutura em apenas dois componentes se deve à presença de reatividade neles. O usuário, ao enviar um comentário, modifica a estrutura visual da postagem, transformando a área de comentários vazia em uma área preenchida com um comentário. O outro componente, *Menu*, é responsável pelos processos de qualificação e classificação:

Figura 20 - Menu da postagem

Ao clicar no símbolo “#”, o estado do componente menu muda:

Figura 21 - Menu da postagem com estado modificado

Após digitar algo na caixa de texto aberta, o estado do componente, mais uma vez, é modificado. A *figura 22* ilustra o método de classificação de um conteúdo.

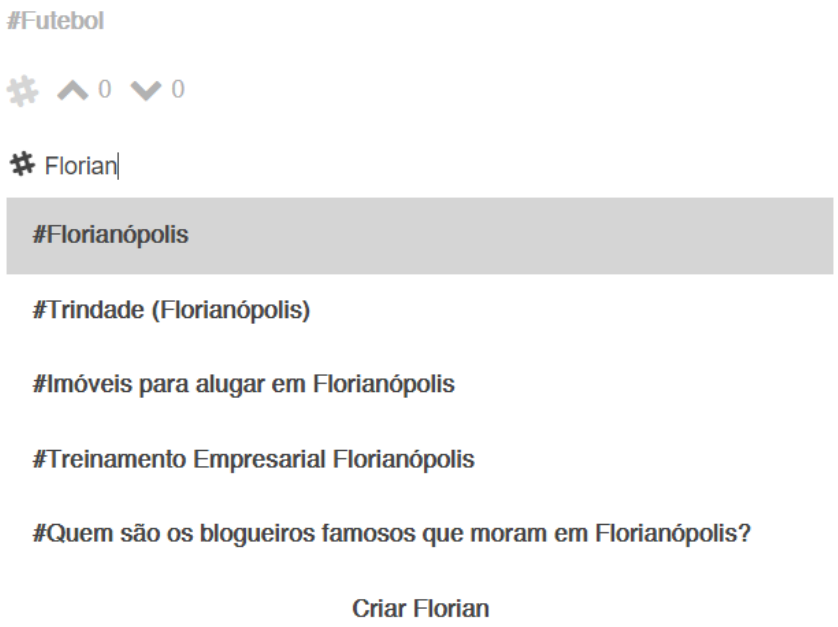

Figura 22 - Classificando um conteúdo

5.2: Canal

O canal é a representação de uma classe de conteúdo e é o elemento que contém as postagens. O canal é a principal forma de visualização da página, sendo o único ponto de acesso do usuário ao conteúdo do site. No Instablah, um canal é chamado de *hashtag,* referente ao elemento "#" presente na cultura digital.

5.2.1: Back-end

Criação da tabela hashtags

```
CREATE TABLE hashtags(
       id serial PRIMARY KEY,
       name text UNIQUE NOT NULL,
       slug text UNIQUE,
       created_at timestamp WITH time zone NOT NULL DEFAULT now(),
       updated_at timestamp WITH time zone NOT NULL DEFAULT now(),
```

```
);
```

Tabela *Hashtags*

Referente à estrutura principal de um canal

Atributo	Tipo
id	serial
name	text
slug	text
created_at	timestamp
updated_at	timestamp

Tabela 5 - Tabela hashtags

Como o canal é definido como uma página do site, é necessário criar uma rota para seu acesso direto. Essa rota é composta pela concatenação do domínio do site com o *slug* do canal, sendo slug uma representação minimizada de seu nome. No caso do canal *Automação*, a rota final seria interpretada como http://beta.instablah.com.br/automacao.

5.2.2: Front-end

Sendo o elemento que contém todos os componentes da aplicação, o canal pode ser representado como a própria página do navegador. A *figura 23* ilustra a estrutura visual de um canal. É possível observar que existe uma lista de postagens na parte central e uma lista à direita com mais conteúdo a ser acessado. A lista central contém todas as postagens que foram criadas naquele canal enquanto a outra contém as melhores postagens desde as últimas 24h. A barra superior compreende o título do site, uma caixa de texto para procurar outros canais e a identificação do próprio usuário: a persona.

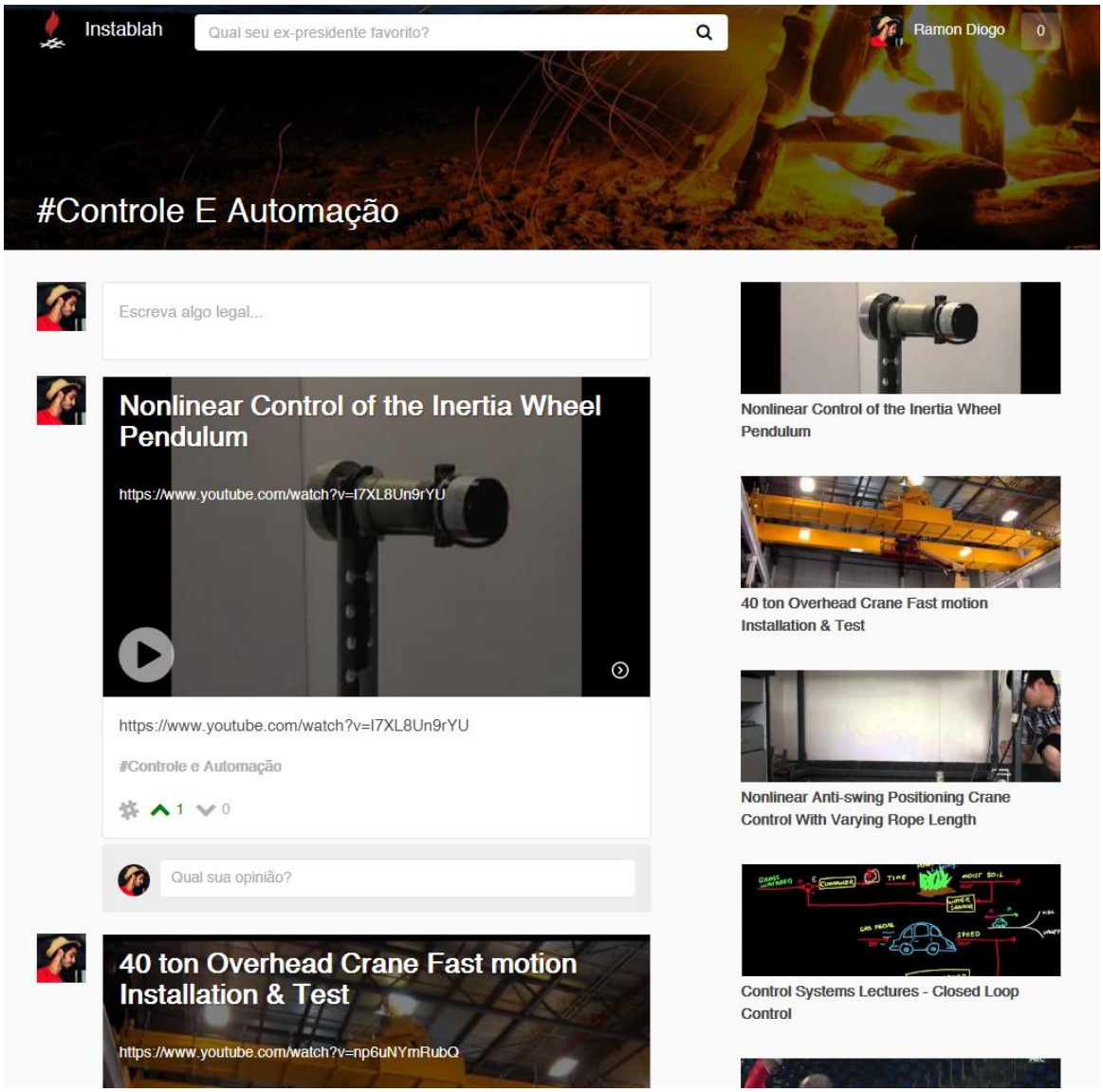

Figura 23 - Visualização da página de um canal

5.3: Comentário

O comentário é a principal interação do usuário com o conteúdo. Sua conotação social é um dos fatores fundamentais de qualquer rede social, já que permite uma comunicação direta entre as pessoas. No caso do Instablah é imprescindível que os usuários comentem em postagens, já que o próprio comentário pode adicionar volume e qualidade à discussão.

5.3.1: Back-end

Criação da tabela comments

```
CREATE TABLE comments(
	id serial PRIMARY KEY,
	text text NOT NULL,
	persona_id integer NOT NULL REFERENCES personas
		ON UPDATE CASCADE ON DELETE CASCADE,
	tagging_id integer NOT NULL REFERENCES taggings
		ON UPDATE CASCADE ON DELETE CASCADE,
	created_at timestamp WITH time zone NOT NULL DEFAULT now(),
	updated_at timestamp WITH time zone NOT NULL DEFAULT now()
	);
```

Tabela *Comments*

Referente aos comentários que podem ser feitos em uma postagem

Atributo	Tipo
id	serial
text	text
persona_id	integer
tagging_id	integer
created_at	timestamp
updated_at	timestamp

Tabela 6 - Tabela comments

5.3.2: Front-end

A camada visual dos comentários foi inspirada no design de um famoso aplicativo de troca de mensagens: o *Whatsapp*. A escolha para esse design se baseou não só no sucesso do produto, mas como na simplicidade que representa. A *figura 24* ilustra a seção de comentários.

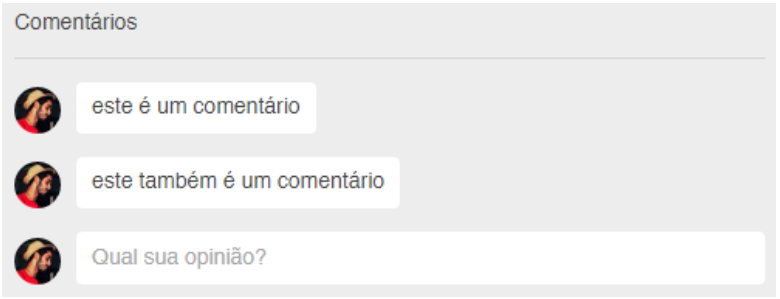

Figura 24 - Visualização de comentários de uma postagem

5.4: Voto

O voto é o meio de definir a qualidade do conteúdo compartilhado. Uma postagem pode receber um voto positivo ou negativo.

5.4.1: Back-end

Criação da tabela votes

```
CREATE TYPE valid_votes AS ENUM (‘like’, ‘dislike’);

CREATE TABLE votes(
	id serial PRIMARY KEY,
	type valid_votes NOT NULL,
	persona_id integer NOT NULL REFERENCES personas
		ON UPDATE CASCADE ON DELETE CASCADE,
	tagging_id integer NOT NULL REFERENCES taggings
		ON UPDATE CASCADE ON DELETE CASCADE,
	comment_id integer NOT NULL REFERECENS comments
		ON UPDATE CASCADE ON DELETE CASCADE,
	created_at timestamp WITH time zone NOT NULL DEFAULT now(),
	updated_at timestamp WITH time zone NOT NULL DEFAULT now()
	);

ALTER TABLE votes ADD CONSTRAINT unique_user_tagging_vote UNIQUE (persona_id, tagging_id);
ALTER TABLE votes ADD CONSTRAINT unique_user_comment_vote UNIQUE (persona_id, comment_id);
```

Tabela *Votes*

Referente aos votos dados a uma postagem

Atributo	Tipo
id	serial
type	valid_votes
persona_id	integer
tagging_id	integer
comment_id	integer
created_at	timestamp
updated_at	timestamp

Tabela 7 - Tabela votes

A tabela de votos possui uma característica diferente das demais: um usuário só pode criar um voto por postagem. Na *query* de criação da tabela há uma parte que adiciona essa limitação. Isso é feito para evitar que uma pessoa possa qualificar um conteúdo várias vezes, o que prejudica a confiança do julgamento. Além disso, o código explicita que só podem existir dois tipos de voto: o positivo e o negativo. Isso ajuda a dar consistência ao banco, prevenindo erros indesejados.

5.4.2: Front-end

A visualização do voto se encontra no menu de uma postagem. As duas opções, negativa e positiva, se encontram um do lado da outra e são representadas por ícones de flechas, como ilustrado na *figura 25*.

Figura 25 - Visualização do voto

Quando o usuário efetua um voto, o estado do componente muda para indicar qual tipo de voto foi realizado (e avisar que a requisição ao servidor foi um sucesso). O novo estado pode ser visualizado na *figura 26*.

Figura 26 - Estado do componente de voto após um voto

5.5: Estruturas secundárias

As estruturas apresentadas até agora podem ser consideradas como atributos primários da aplicação, afinal são os responsáveis pelo modelo básico da ideia de uma rede social. Por terem um modo de visualização, são mais fáceis de entender. Há ainda estruturas secundárias que possibilitam comportamentos mais complexos para a aplicação e que não são visíveis ao usuário. Essas estruturas são apresentadas a seguir.

5.5.1: Tagging

Para que as postagens possam aparecer nos canais, é preciso criar um relacionamento, no banco de dados, entre as duas entidades. Uma postagem pode pertencer a vários canais – como se tivesse várias classificações – enquanto um canal possui várias postagens. Para criar essa relação, foi criada a tabela *taggings* que carrega, em sua tabela, o id de uma postagem e o id de um canal, sendo que não podem existir relacionamentos repetidos (onde a mesma postagem se encontra duas vezes no mesmo canal).

Criação da tabela taggings

```
CREATE TABLE taggings(
	id serial PRIMARY KEY,
	post_id integer NOT NULL REFERENCES posts
		ON UPDATE CASCADE ON DELETE CASCADE,
	hashtag_id integer NOT NULL REFERENCES hashtags
		ON UPDATE CASCADE ON DELETE CASCADE,
	persona_id integer NOT NULL REFERECENS peronas
		ON UPDATE CASCADE ON DELETE CASCADE,
	created_at timestamp WITH time zone NOT NULL DEFAULT now(),
```

```
	updated_at timestamp WITH time zone NOT NULL DEFAULT now()
	);

ALTER TABLE taggings ADD CONSTRAINT unique_tagging UNIQUE (post_id, hashtag_id);
```

Tabela *Taggings*

Referente ao relacionamento entre postagens e canais

Atributo	Tipo
id	serial
post_id	integer
hashtag_id	integer
persona_id	integer
created_at	timestamp
updated_at	timestamp

Tabela 8 - Tabela taggings

5.5.2: Usuário

A tabela de usuários serve como um identificador da origem do usuário. Ela representa a pessoa real (física) usando a aplicação. Suas colunas têm todas as variáveis necessárias para se manter uma sessão no site. Além disso, possui informação sobre autorização para realizar certas ações, podendo identificar o usuário como membro administrador ou um funcionário da empresa.

Criação da tabela users

```
CREATE TABLE users(
	id serial NOT NULL PRIMARY KEY,
	password varchar(128) NULL,
	last_login timestamp WITH time zone NULL,
	is_superuser boolean NOT NULL,
	email varchar(75) NOT NULL,
	is_staff boolean NOT NULL,
	is_active boolean NOT NULL,
	fb_name TEXT NULL,
```

```
    fb_token TEXT NULL,
    fb_uid TEXT UNIQUE,
    default_persona_id integer NULL REFERENCES personas,
    date_joined timestamp WITH time zone NOT NULL
    );
```

Tabela *Users*

Referente aos dados do usuário

Atributo	Tipo
id	serial
password	varchar(128)
last_login	timestamp
is_superuser	boolean
email	varchar(75)
is_staff	boolean
is_active	boolean
fb_name	text
fb_token	text
fb_uid	text
default_persona_id	integer
date_joined	timessetamp

Tabela 9 - Tabela users

5.5.3: Persona

Um dos atributos futuros do site será a criação de personagens – ou pseudônimos – com intuito de incentivar a participação em diversos canais. Pensando nisso, foi desenvolvida a tabela personas, que serve como uma apresentação do usuário para o restante da rede. Uma persona possui nome e imagem, além de ser a entidade que realiza as ações de postar, comentar, votar e classificar.

Criação da tabela personas

```
CREATE TABLE personas(
	id serial PRIMARY KEY,
	name text NOT NULL,
	slug text UNIQUE,
	image text NOT NULL,
	last_seen bigint,
	created_at timestamp WITH time zone NOT NULL DEFAULT now(),
	updated_at timestamp WITH time zone NOT NULL DEFAULT now()
	);
```

Tabela *Personas*

Referente à apresentação do usuário perante a rede

Atributo	Tipo
id	serial
name	text
slug	text
image	text
last_seen	bigint
created_at	timestamp
updated_at	timestamp

Tabela 10 - Tabela personas

5.6: Requisição das postagens

Ao carregar a página de um canal, existem duas formas de requisitar as postagens do servidor: trazê-las em ordem de criação ou trazê-las ordenadas pela diferença entre número de votos positivos e negativos (ordenadas por sua qualidade). Devido aos relacionamentos presentes e à complexa estrutura da rede, a dificuldade presente em montar as queries foi grande e, por isso, considerou-se pertinente citá-las neste documento.

5.6.1: Últimas postagens

Visão para requerer as últimas postagens de um canal

```
CREATE VIEW hashtag_posts_view AS
    SELECT
        p.id,
        p.type,
        h.slug as hashtag_slug,
        p.slug,
        row_to_json(personas) as persona,
        t.id as tagging_id,
        t.created_at,
        (select case
            when post_links.id is not null then row_to_json(post_links)
            when post_images.id is not null then row_to_json(post_images)
            when post_texts.id is not null then row_to_json(post_texts)
            when post_videos.id is not null then row_to_json(post_videos)
        end) as feature
    FROM
        posts p
    INNER JOIN
        personas ON (personas.id = p.persona_id)
    INNER JOIN
        taggings t ON (p.id = t.post_id)
    INNER JOIN
        hashtags h ON (t.hashtag_id = h.id)
    LEFT JOIN
        post_links ON p.id = post_links.post_id
    LEFT JOIN
        post_images ON p.id = post_images.post_id
    LEFT JOIN
        post_texts ON p.id = post_texts.post_id
    LEFT JOIN
        post_videos ON p.id = post_videos.post_id
    ORDER BY
        t.created_at DESC, t.id DESC
```

5.6.2: Melhores postagens

Visão para requerer as melhores postagens de um canal

```
CREATE VIEW best_posts_view AS
    SELECT
        *
    FROM(SELECT
        p.id,
        p.type,
        h.slug as hashtag_slug,
        p.slug,
        row_to_json(personas) as persona,
        t.id as tagging_id,
        t.created_at,
        SUM(CASE WHEN v.type = 'like' THEN 1 ELSE 0 END
          - CASE WHEN v.type = 'dislike' THEN 1 ELSE 0 END) AS total_likes,
        (select case
            when post_links.id is not null then row_to_json(post_links)
            when post_images.id is not null then row_to_json(post_images)
            when post_texts.id is not null then row_to_json(post_texts)
            when post_videos.id is not null then row_to_json(post_videos)
        end) as feature
    FROM
        posts p
    INNER JOIN
        personas ON (personas.id = p.persona_id)
    INNER JOIN
        taggings t ON (p.id = t.post_id)
    INNER JOIN
        hashtags h ON (t.hashtag_id = h.id)
    LEFT JOIN
        votes v ON (v.tagging_id = t.id)
    LEFT JOIN
        post_links ON p.id = post_links.post_id
    LEFT JOIN
        post_images ON p.id = post_images.post_id
    LEFT JOIN
        post_texts ON p.id = post_texts.post_id
    LEFT JOIN
```

```
        post_videos ON p.id = post_videos.post_id
    GROUP BY p.id,
         h.slug,
         post_links.id,
         post_images.id,
         post_texts.id,
         post_videos.id,
         personas.id,
         t.id) AS posts
    ORDER BY
        posts.total_likes DESC, tagging_id DESC
```

5.6.3: Melhores postagens desde as últimas 24 horas

Visão para requerer as melhores postagens de um canal desde as últimas 24 horas

```
CREATE VIEW relateds_view AS
    SELECT
        *
    FROM (SELECT DISTINCT ON (p.id)
        p.id,
        h.slug AS hashtag_slug,
        p.slug,
        SUM(CASE WHEN v.type = 'like' AND v.created_at > current_timestamp -
interval '24 hour' THEN 1 ELSE 0 END
          - CASE WHEN v.type = 'dislike' AND v.created_at > current_timestamp -
interval '24 hour' THEN 1 ELSE 0 END) AS total_likes,
        (SELECT
            CASE
                WHEN post_links.id IS NOT NULL THEN row_to_json(post_links)
                WHEN post_images.id IS NOT NULL THEN row_to_json(post_images)
                WHEN post_texts.id IS NOT NULL THEN row_to_json(post_texts)
                WHEN post_videos.id IS NOT NULL THEN row_to_json(post_videos)
            END) AS feature
    FROM
        posts p
    INNER JOIN personas
        ON (personas.id = p.persona_id)
    INNER JOIN taggings t
        ON (p.id = t.post_id)
```

```
    INNER JOIN hashtags h
        ON (t.hashtag_id = h.id)
    LEFT JOIN votes v
        ON (v.tagging_id = t.id)
    LEFT JOIN post_links
        ON p.id = post_links.post_id
    LEFT JOIN post_images
        ON p.id = post_images.post_id
    LEFT JOIN post_texts
        ON p.id = post_texts.post_id
    LEFT JOIN post_videos
        ON p.id = post_videos.post_id
    WHERE
        p.slug IS NOT NULL
    GROUP BY p.id,
         h.slug,
         post_links.id,
         post_images.id,
         post_texts.id,
         post_videos.id) AS relateds
    WHERE
        (feature::json->'img' IS NOT NULL OR feature::json->'source_img' IS NOT
NULL)
    ORDER BY relateds.total_likes DESC, id DESC
    LIMIT 30
```

5.7: Componentes

Como citado antes, a camada de visualização da página é estruturada por meio de componentes que possuem um escopo próprio. Tais componentes se comunicam com uma árvore de dados para garantir que a interação do usuário seja previsível e controlada. A seguir são listados e mostrados os componentes da página.

5.7.1: Postagem e comentários

Como já mostrado antes, as postagens são componentes responsáveis por mostrar o conteúdo ao usuário. Elas contêm ainda dois componentes: o menu da postagem e a área de comentários.

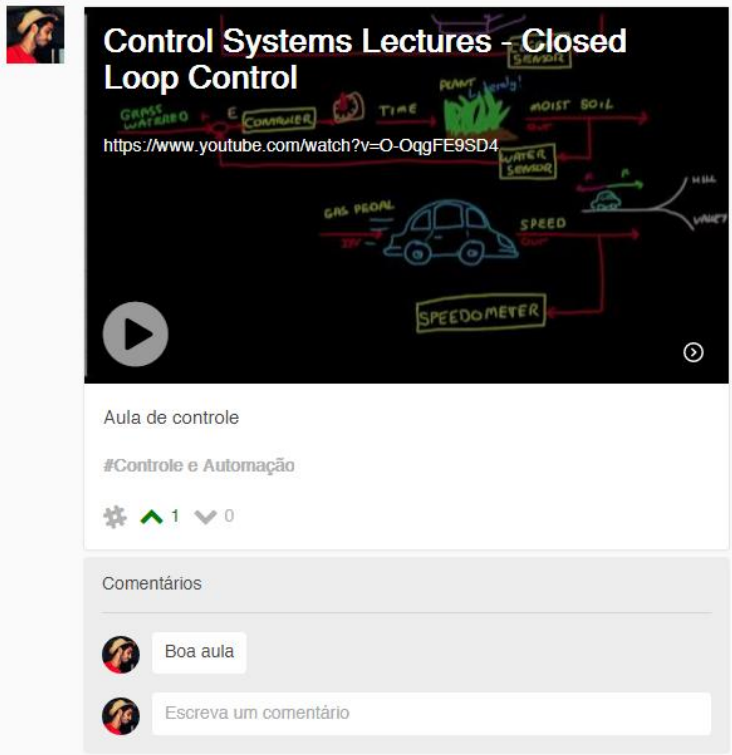

Figura 27 - Componente da postagem

5.7.2: Novas postagens

Há um componente responsável apenas por mostrar novas postagens criadas pelo usuário, garantindo assim que a cada nova postagem criada sejam inseridos seus dados no topo do canal, garantindo um maior controle sobre essa estrutura.

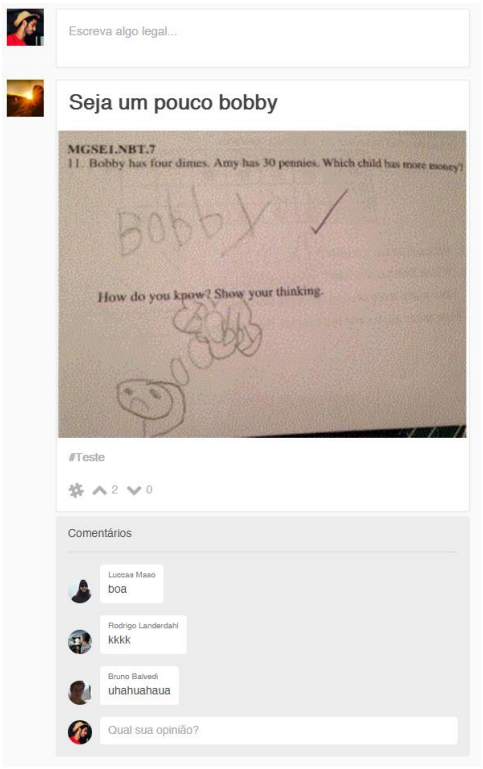

Figura 28 - Primeira etapa de uma postagem, quando a página acabou de ser carregada

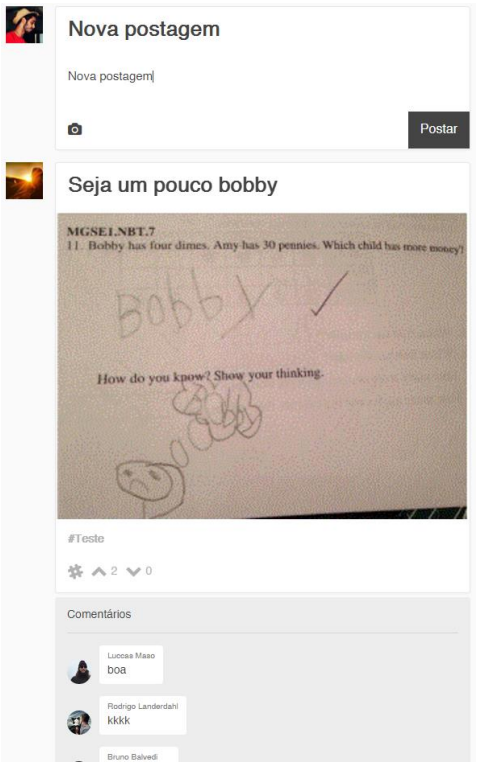

Figura 29 - Segunda etapa de uma postagem, quando o usuário está digitando

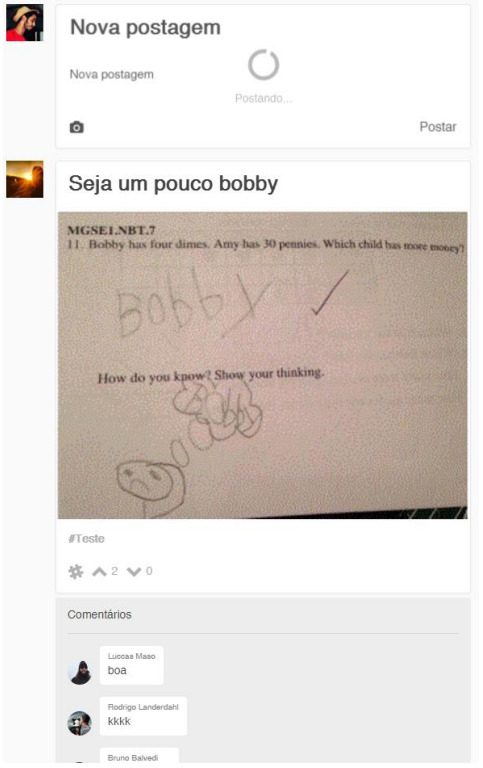

Figura 30 - Terceira etapa, quando o usuário envia a postagem ao servidor

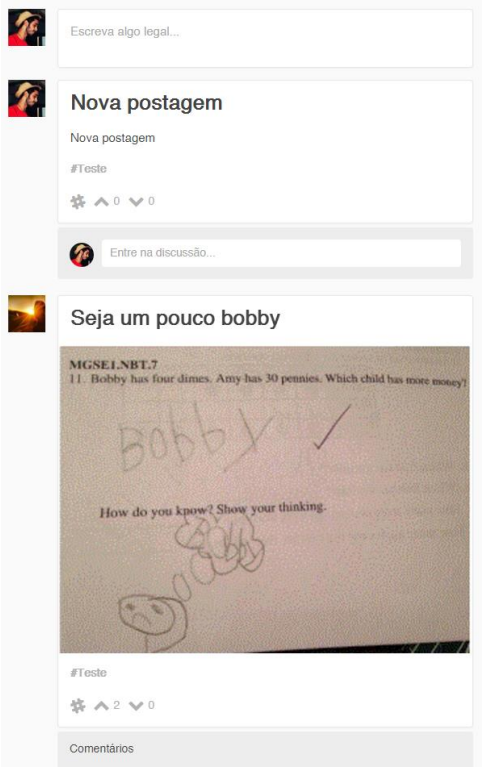

Figura 31 - Quarta e última etapa, quando o componente adiciona a postagem à página, sem recarregá-la

5.7.3: Antigas postagens

Toda vez que o usuário entra em um canal, são carregadas, no máximo, dez postagens. O objetivo é evitar uma lentidão na resposta do servidor, limitando o número de entidades a retornar. Se o usuário quiser ver mais conteúdo, é necessário ir até a metade da página. Ao captar esse evento, o componente responsável por carregar mais postagens sabe que tem que agir e assim mais dez entidades são carregadas.

5.7.4: Entrada de postagens

Esse componente é responsável por criar conteúdo. Por ser responsável pela criação de 4 tipos de postagens, sua complexidade se sobressai aos demais componentes. A *figura 32* mostra esse componente em seu estado inicial.

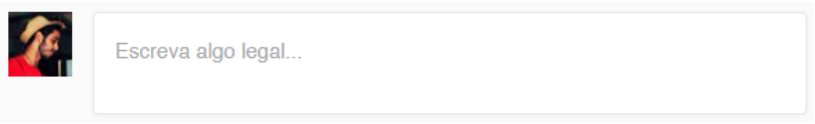

Figura 32 - Entrada de postagens

Ao ser clicado, o componente entra em um estado de edição, expandindo e habilitando novas ações, como enviar uma imagem do computador, inserir um título e efetuar o envio da postagem. A *figura 33* ilustra esse estado.

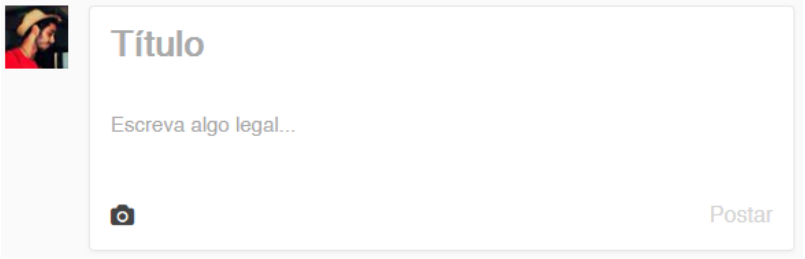

Figura 33 - Entrada de postagens estendida

Caso o usuário queira compartilhar conteúdo de outros sites, existe a opção de copiar o endereço desejado e colar na área de postagem. Ao fazer isso, um

script acessa no site indicado, copia o principal conteúdo e monta uma prévia da página com imagem, texto e título.

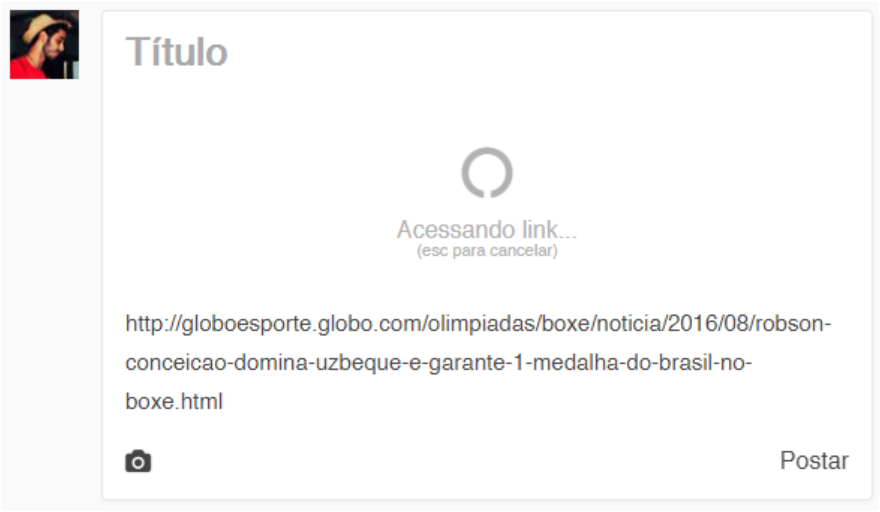

Figura 34 - Quando o usuário cola um endereço, um script começa a buscar informações da página

Figura 35 - Sistema termina de buscar informações e monta uma prévia

As estruturas citadas nesse capítulo completam a primeira versão do Instablah, restando para o projeto apenas a análise dos resultados.

5.7.5: Melhores postagens

Uma área lateral é reservada apenas para apresentar o melhor conteúdo postado no site nas últimas 24h. Essa área inteira é tratada como um componente, mesmo não possuindo nenhum tipo em interação em tempo real. Isso ocorre devido à precaução da equipe a futuras implementações de atributos nessa área.

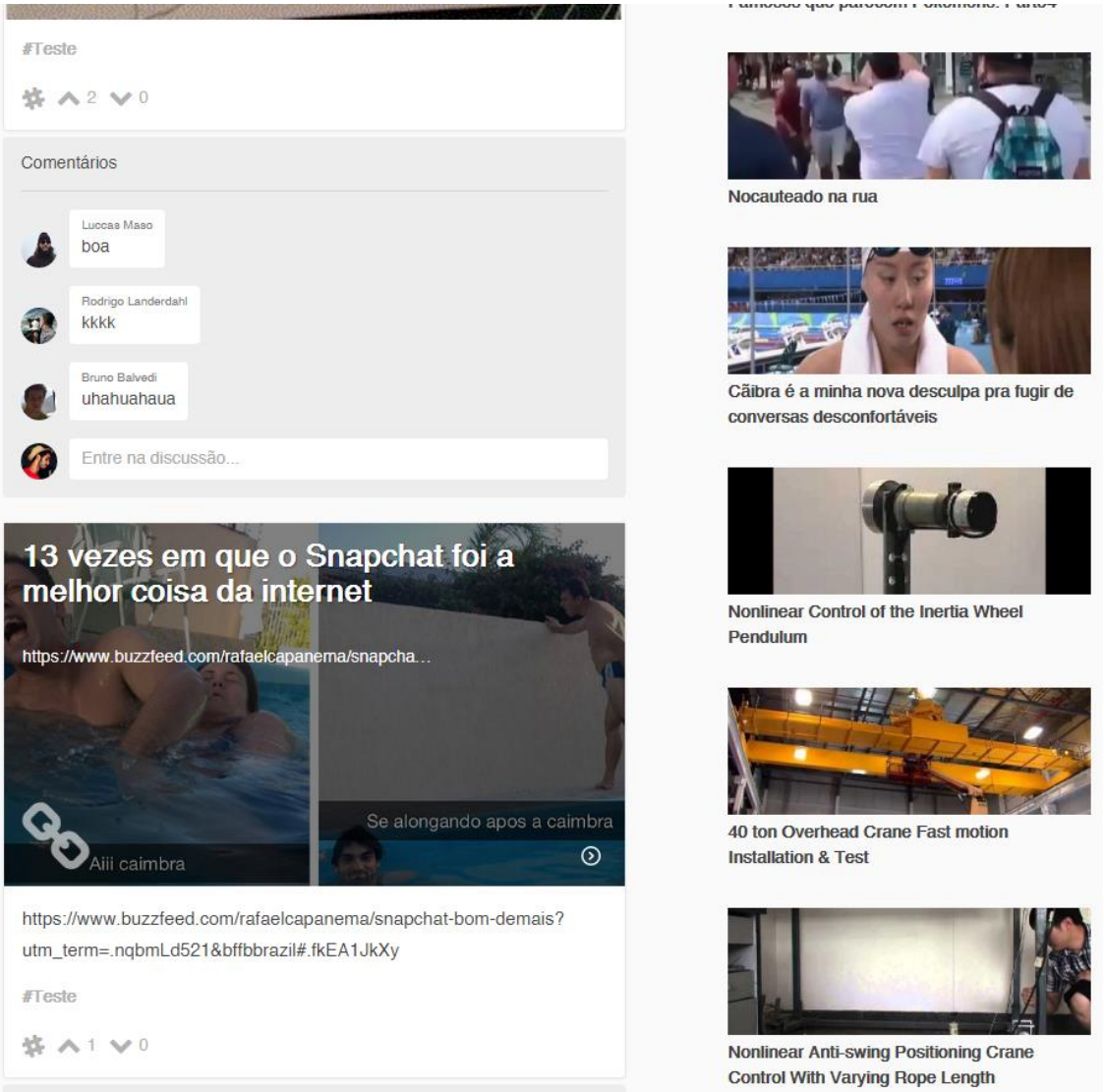

Figura 36 - Componente das melhores postagens do dia (à direita)

Capítulo 6: Resultados

Com a primeira versão pronta e *online*, foram realizados testes de divulgação. A equipe, ciente de que não possui uma versão final da rede, decidiu que não seria bom, para a imagem da rede, divulgar o projeto na grande mídia, limitando-se apenas a pequenos grupos de pessoas. Essa é uma prática comum adotada por pequenas empresas, visando obter um maior conhecimento sobre a usabilidade de seu produto. Com isso são descobertos bugs e gargalos na aplicação, contribuindo para uma rápida, e controlada, melhora de suas características e interações.

Do dia 31 de julho até o dia 10 de agosto, foram divulgadas postagens do Instablah em pequenos grupos do *Facebook*. O resultado é ilustrado na *figura 37*.

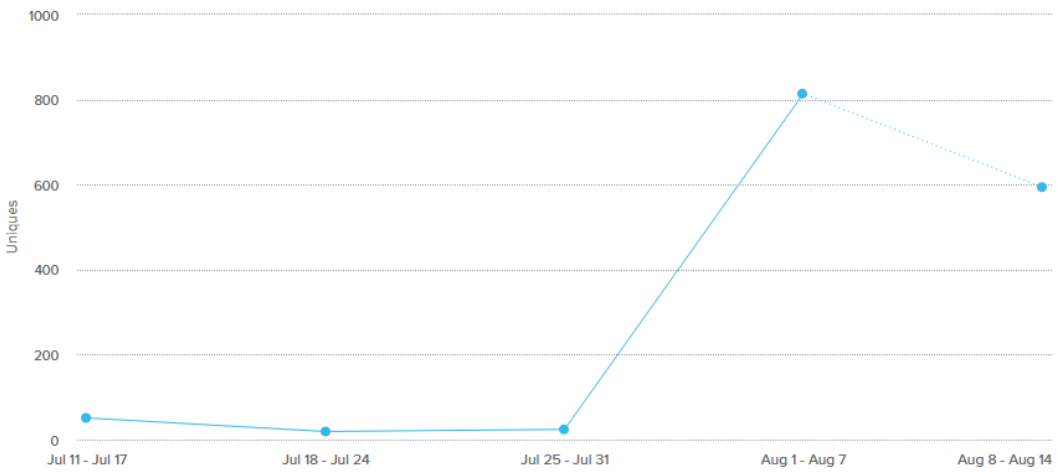

Figura 37 – Número de pessoas que entraram no site por semana

Conforme observado no gráfico, cerca de 1400 pessoas visitaram o site. Mais testes devem ser realizados para garantir se esse é um bom ou mau resultado, porém, baseado na experiência da equipe, o número agradou, considerando o pequeno esforço despendido.

Uma boa métrica utilizada para garantir que um produto esteja dando certo é a retenção. Após a primeira visita, espera-se que o usuário volte a visitar o site e, em uma de suas visitas, torne-se um membro da rede. Existem várias maneiras de

medir isso. O gráfico da *figura 38* ilustra quantas pessoas entraram no site em dois dias diferentes de uma semana e pode ser considerado um bom indicativo de que as pessoas estão sendo retidas.

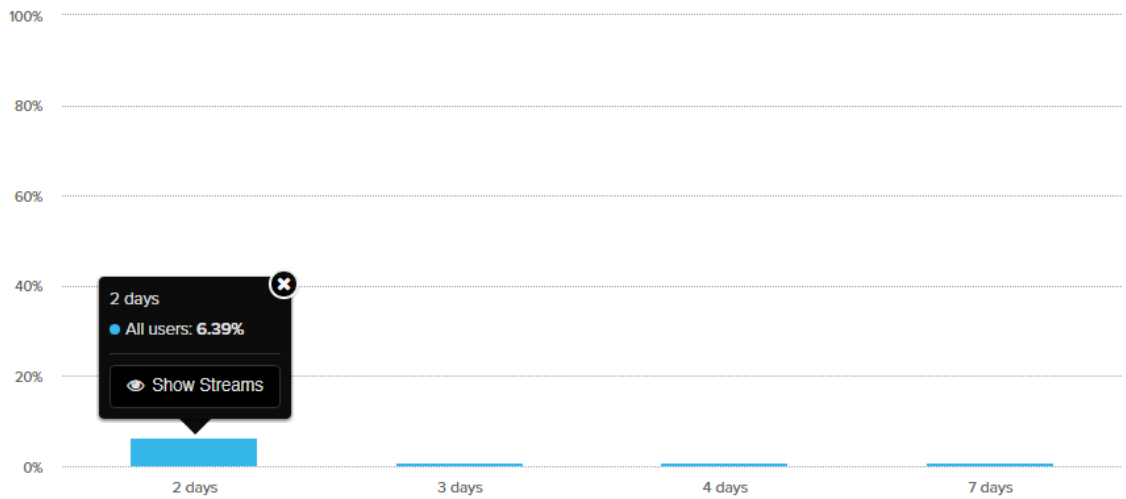

Figura 38 - Número de pessoas que visitaram o site em dois dias diferentes de uma mesma semana

O gráfico considera as 814 pessoas que visitaram o site na primeira semana e mostra que 6.39% (52 pessoas) do total voltaram uma segunda vez durante a mesma semana. Devido à pouca quantidade de visitantes e o pouco tempo de lançamento fica difícil saber se o resultado é positivo ou negativo. Ainda são necessários mais testes para saber se o padrão se repete ou se foi algo pontual.

Ainda considerando a análise do comportamento do usuário, pode-se obter o número de pessoas que entraram no site e navegaram por pelo menos mais uma página da aplicação. A figura 39 ilustra o resultado para duas semanas em que foram realizados os testes.

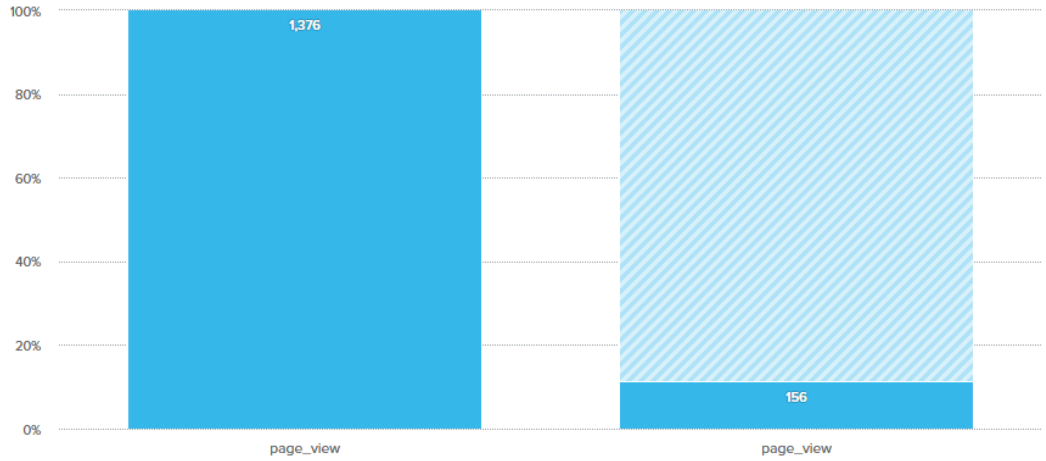

Figura 39 - Funil de profundidade de navegação pelo site

Todos os resultados obtidos nessas duas semanas mostram que o site possui capacidade de crescer. O objetivo inicial o projeto é ajudar as pessoas a encontrar conteúdos de seu interesse sem serem bombardeadas de dados supérfluos. Diminuir a sobrecarga de informação para as pessoas é o que motiva o desenvolvimento da rede social Instablah. Infelizmente, devido ao curto espaço de tempo para a produção deste documento, dados concretos que ajudassem na resolução desse problema não puderam ser levantados.

Capítulo 7: Conclusões e Perspectivas

Produzir uma rede social se mostrou uma tarefa muito difícil. É necessário um extenso tempo de estudo para escolher metodologias e tecnologias que se encaixem nos requisitos do sistema. Deve-se lembrar que esse sistema pode ser acessado por milhões de pessoas ao mesmo tempo, o que requer um alto nível de segurança e uma estrutura que permita uma rápida detecção de erros, afim de evitar transtorno para os usuários.

O desenvolvimento do Instablah me permitiu trabalhar com tecnologias de ponta, principalmente a linguagem Erlang, tão temida no mundo da programação. Novos horizontes se abriram e a dificuldade enfrentada no começo do projeto (no período de aprendizagem da linguagem) transformou-se em um desenvolvimento fluido e eficaz ao final. Além disso, o estudo da arquitetura *Flux* ajudou a otimizar o tempo de produção e diminuir o número de erros no código.

Para o futuro, não tão distante, a rede espera poder contar com cada vez mais usuários e, com isso, ir aprendendo qual caminho tomar e quais atributos desenvolver. No período de conclusão deste documento, já está sendo produzida uma versão móvel do site, possibilitando uma melhor visualização do site no celular.

Capítulo 8: Bibliografia

[1] GOOGLE (Empresa). *Enabling Trade in the Era of Information Technologies: Breaking Down Barriers to the Free Flow of Information*. Disponível em https://static.googleusercontent.com/media/www.google.com/en//googleblogs/pdfs/trade_free_flow_of_information.pdf Acesso em 07/08/2016.

[2] ORG. FOR ECON. COOPERATION & DEV. *Broadband and the Economy: Ministerial Background Report*. Disponível em https://www.oecd.org/sti/40781696.pdf Acesso em 07/08/2016.

[3] BOYD, DANA M.; ELLISON, NICOLE B. *Social Network Sites: Definition, History, and Scholarship*. Disponível em http://www.danah.org/papers/JCMCIntro.pdf Acesso em 07/08/2016.

[4] STATISTA (Empresa). *Global Social Networks Ranked by Number of Users*. Disponível em http://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/ Acesso em 07/08/2016.

[5] ZEPHORIA DIGITAL MARKETING (Empresa). *The Top 20 Valuable Facebook Statistics*. Disponível em https://zephoria.com/top-15-valuable-facebook-statistics/ Acesso em 07/08/2016.

[6] EPPLER, M.; MENGIS, J. (2002). *The Concept of Information Overload: A review of literature from organization science, marketing, accounting, MIS, and related disciplines*. MCM Research Paper.

[7] INTERNET LIVE STATS (Empresa). *Internet Users*. Disponível em http://www.internetlivestats.com/internet-users/ Acesso em 07/08/2016.

[8] LIN, XINYE; XIA, MINGYUAN; LIU, XUE. *Does "Like" Really Mean Like? A Study of the Facebook Fake Like Phenomenon and an Efficient Countermeasure*. Montreal, 2014.

[9] ZESTY. *Why Do People Click 3 Billion Facebook Like Buttons Each Day*? Disponível em https://www.simplyzesty.com/blog/article/july-2010/why-do-people-click-3-billion-facebook-like-buttons-each-day Acesso em 10/08/2016.

[10] CESARINI, FRANCESCO; THOMPSON, SIMON. *Erlang Programming, A Concurrent Approach to Software Development*. California, 2009. 1 p.

[11] CESARINI, FRANCESCO; THOMPSON, SIMON. *Erlang Programming, A Concurrent Approach to Software Development*. California, 2009. 2 p.

[12] NYSTRÖM, J. H.; TRINDER, P. W.; KING, D. J. *Concurrency and Computation: Practice & Experience*, 2008.

[13] MILLER, EVAN. Why I Program Erlang. Disponível em http://www.evanmiller.org/why-i-program-in-erlang.html Acesso em 12/08/2016.

[14] MILLER, EVAN. *Chicago Boss: A Rough Introduction*. Disponível em http://chicagoboss.org/tutorial.pdf Acesso em 10/08/2016.

[15] YOU, EVAN. *Vue.js Overview*. Disponível em https://vuejs.org/guide/overview.html Acesso em 10/08/2016.

[16] YOU, EVAN. *What is Vuex?* Disponível em http://vuex.vuejs.org/en/intro.html Acesso em 10/08/2016.

[17] OBE, REGINA; HSU, LEO. PostgreSQL Up & Running. A Practical Guide to the Advanced Open Source Database, 2nd Edition. California, 2015.

[18] CLARK, LIN. A Cartoon Guide to Flux. Disponível em https://code-cartoons.com/a-cartoon-guide-to-flux-6157355ab207 Acesso em 11/08/2016.