

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS

LUCAS HENRIQUE DE ABREU

DESENVOLVIMENTO DE UMA APLICAÇÃO DE REDE SOCIAL CORPORATIVA

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2017

LUCAS HENRIQUE DE ABREU

**DESENVOLVIMENTO DE UMA APLICAÇÃO DE REDE SOCIAL
CORPORATIVA**

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Vinicius Pegorini

PATO BRANCO
2017



Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento
de Sistemas



TERMO DE APROVAÇÃO
TRABALHO DE CONCLUSÃO DE CURSO
DESENVOLVIMENTO DE UMA APLICAÇÃO DE REDE SOCIAL
CORPORATIVA

POR

LUCAS HENRIQUE DE ABREU

Este trabalho de conclusão de curso foi apresentado no dia 11 de dezembro de 2017, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Prof MSc. Vinicius Pegorini
Orientadora

Profª Drª Beatriz Terezinha Borsoi

Prof. Dr. Ives Rene Venturini Pola

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em
Análise e Desenvolvimento de Sistemas

Profª Drª Beatriz Terezinha Borsoi
Responsável pela Atividade de Trabalho de
Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

ABREU, Lucas Henrique. Desenvolvimento de uma aplicação de redes sociais corporativas. 2017. 72f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

O número de empresas no Brasil é crescente e o volume de dados que elas produzem e utilizam é bastante grande. As empresas têm a necessidade de realizar comunicação interna, trocar de arquivos, disseminar conhecimento, agendar reuniões e eventos, reservar de salas e materiais, entre outros. E para isso elas acabam utilizando soluções distintas ou utilizando, basicamente, seu servidor de *e-mail* para todas essas atividades. Um sistema que atenda essas necessidades e outras no contexto empresarial se assemelha a uma rede social como o Facebook. Pelo amplo uso das redes sociais e familiarização do uso dessas redes pelas pessoas, o desenvolvimento de uma aplicação *web* baseada em uma rede social tem a vantagem de conhecimento de uso. A utilização de uma aplicação nesse formato fornece uma visão bastante ampla ao gerente de Recursos Humanos a partir de uma lapidação dos dados gerados pelos usuários e colaboradores da empresa. Esses dados podem ser apresentados no formato de *dashboards*, com preferências e avaliação do nível de satisfação dos funcionários da empresa e do local de trabalho. Considerando que as organizações visam uma melhor comunicação entre seus colaboradores e a necessidade de centralizar informações, arquivos e fluxo de trabalho, neste projeto foi desenvolvido um software que possui funcionalidades relacionadas ao gerenciamento de perfil do usuário, linha do tempo, bate-papo, agendamento de eventos e criação de grupos. Além de fornecer dados para o administrador que possibilitem uma análise de cada usuário e dos grupos, visando melhoria na comunicação e agilidade no fluxo de trabalho.

Palavras-chave: Redes Sociais. Ambiente Corporativo. JavaScript.

ABSTRACT

ABREU, Lucas Henrique. Development of an application of corporate social networks. 2017. 72f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

The number of companies in Brazil is increasing, as well as the volume of data that these produce and use. The companies have the need of internal communication, exchange of files, dissemination of knowledge, scheduling of meetings and events, reservation of rooms and materials, among others, however, to meet their demands, they end up using distinct solutions or basically using their mail server for all of these activities. Therefore, understanding that organizations seek better communication among their employees and have the need to centralize information, files and workflow, this project was develop a software that has features related to the management of user profile, timeline, chat, event scheduling, and group creation. The application meets all the needs presented and is developed as a web application based on a social network, such as Facebook, because the widespread use of social networks and familiarization of the use of such networks by people facilitates the adaptation of this system. The application in this format provides a very broad vision to the Human Resources Manager from the manipulation of the data produced by the users and employees of the company. This data can be presented in the dashboard format, with preferences and assessment of the level of employee satisfaction with the company and the workplace. In addition, the application provides data to the administrator that enables an analysis of each user and groups, aiming at improving communication and agility in the workflow.

Keywords: Social Networks. Corporate Environment. JavaScript.

LISTA DE FIGURAS

Figura 1 - Diagrama de casos de casos de uso.....	27
Figura 2 - Diagrama de entidade e relacionamento.....	29
Figura 3 - Cadastro de conta	31
Figura 4 - E-mail de confirmação da conta.....	32
Figura 5 - Tela de boas vindas	33
Figura 6 - Tela de login.....	33
Figura 7 - Tela de postagens	34
Figura 8 - Janela de criação de postagens.....	35
Figura 9 - Aba de convites	35
Figura 10 - Convite	36
Figura 11 - Pré-cadastro do usuário.....	37
Figura 12 - Perfil - dados pessoais	38
Figura 13 - Perfil - publicações.....	39
Figura 14 - Perfil - eventos do usuário	40
Figura 15 - Listagem de eventos	41
Figura 16 - Criação de eventos	42
Figura 17 - Visão detalhada do evento	43
Figura 18 - Visão principal da aplicação.....	44

LISTA DE QUADROS

Quadro 1 - Ferramentas e tecnologias	19
Quadro 2 - Listagem dos requisitos funcionais do sistema.....	26
Quadro 3 - Listagem dos requisitos não funcionais do sistema.....	27

LISTAGENS DE CÓDIGO

Listagem 1 - Estrutura de diretórios da API	45
Listagem 2 - Padrão REST de requisições	46
Listagem 3 - configuração de policieis.....	47
Listagem 4 - policy responsável por validar o token do usuário	48
Listagem 5 - AuthController	50
Listagem 6 - Função que busca o usuário logado	51
Listagem 7 - Model de usuário	52
Listagem 8 - Função sobrescrita para criar usuário.....	53
Listagem 9 - Estrutura de diretórios do cliente	55
Listagem 10 - exemplo do módulo principal	57
Listagem 11 - Definição das rotas.....	58
Listagem 12 - Formulário de login	59
Listagem 13 - Método de login no modelo de controle	60
Listagem 14 - Método de login no serviço de autenticação.....	61
Listagem 15 - Serviço para realizar requisições no servidor.....	63
Listagem 16 - MainTimeLineComponent	64
Listagem 17 - Importação de módulos externos	65
Listagem 18 - Declaração do service do Dialog.....	65
Listagem 19 - Evento de click	66
Listagem 20 - Método de criação do dialog.....	66
Listagem 21 - DialogPostComponent	67
Listagem 22 - Template dialog-post.component.html.....	68
Listagem 23 - Serviço de interceptação de requisições http	69

LISTA DE SIGLAS

<i>ACID</i>	<i>Atomicidade, Consistência, Isolamento e Durabilidade</i>
<i>API</i>	<i>Application Programming Interface</i>
<i>CRUD</i>	<i>Create, Read, Update and Delete</i>
<i>HTML</i>	<i>HyperText Markup Language</i>
<i>HTTP</i>	<i>HyperText Transfer Protocol</i>
<i>JSON</i>	<i>JavaScript Object Notation</i>
<i>MVC</i>	<i>Model-View-Controller</i>
<i>MVP</i>	<i>Mínimo Produto Viável</i>
<i>MVR</i>	<i>Model-View-Routes</i>
<i>NoSQL</i>	<i>Not Only SQL</i>
<i>SQL</i>	<i>Structured Query Language</i>
<i>ORM</i>	<i>Object Relational Mapper</i>
<i>REST</i>	<i>Representational State Transfer</i>
<i>URL</i>	<i>Uniform Resource Locator</i>

SUMÁRIO

1 INTRODUÇÃO.....	10
1.1 CONSIDERAÇÕES INICIAIS	10
1.2 OBJETIVOS.....	11
1.2.1 Objetivo Geral.....	11
1.2.2 Objetivos Específicos.....	11
1.3 JUSTIFICATIVA	11
1.4 ESTRUTURA DO TRABALHO	12
2 REFERENCIAL TEÓRICO	13
2.1 REDES	13
2.2 REDES SOCIAIS	14
2.3 BIG DATA NAS REDES SOCIAIS	15
2.4 REDES SOCIAIS CORPORATIVAS.....	17
3 MATERIAIS E MÉTODO	19
3.1 MATERIAIS	19
3.1.2 JavaScript.....	20
3.1.3 TypeScript.....	20
3.1.4 SailsJS.....	20
3.1.5 Node.JS.....	20
3.1.6 Express	21
3.1.7 Angular.....	22
3.1.8 MongoDB.....	22
3.2 MÉTODO.....	23
4 RESULTADO	24
4.1 ESCOPO DO SISTEMA.....	24
4.2 MODELAGEM DO SISTEMA	25
4.3 APRESENTAÇÃO DO SISTEMA	30
4.4 IMPLEMENTAÇÃO DO SISTEMA	44
4.4.1 Servidor	44
4.4.2 Cliente.....	53
5 CONCLUSÃO.....	70
REFERÊNCIAS.....	71

1 INTRODUÇÃO

Este capítulo apresenta a introdução do trabalho que contém as considerações iniciais, objetivos e a justificativa. Por fim, está a apresentação dos outros capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

No primeiro semestre de 2016, o número de novas empresas no país chegou a 1.020.740 (ALBUQUERQUE, 2016). Esse número é crescente, especialmente se incluído o modelo *startups* que já vem sendo difundido desde a década de 1990, mas só ganhou força no Brasil em 2010 e 2011. Para Alves (2013) o modelo de *startups* possui os seguintes atributos:

Startup é uma empresa iniciante com um modelo inovador, que atua em um cenário de incertezas e busca o maior lucro possível em um menor tempo possível. As *startups* atraem capital de risco, devido ao cenário de incertezas, a maioria possui base tecnológica inovadora vinculada à internet (ALVES, 2013, p.10).

O meio de comunicação empresarial é muito importante para que um processo funcione bem. Segundo Bahia (2013), comunicação é um conjunto de métodos, técnicas, recursos e meios pelos quais a empresa se dirige ao público interno.

O uso das redes sociais é um exemplo perfeito da evolução que a *web* teve no surgimento da chamada Web 2.0. Com democratização da forma de distribuição de conteúdo e facilitação de comunicação, surgem a cada dia as mais variadas formas de distribuir informação. Um exemplo disso é o uso do recurso chamado criação de histórias, que são fotos, vídeos ou pequenos textos do cotidiano dos usuários, nem sempre isso tem algum valor, porém esse tipo de conteúdo alcança um grande número de usuários. É evidente que empresas não pagariam pessoas que tem uma quantidade de seguidores muito grande, sem que elas exerçam influência nos seus seguidores.

A percepção das vantagens na comunicação que as redes sociais virtuais promovem

nas empresas, não relacionado ao meio comercial e sim na comunicação interna, ainda não é muito explorado. A comunicação é realizada de forma distribuída, em várias fontes de dados, causando dificuldade na mineração destes dados, para gerar indicadores úteis à gestão da organização. A proposta deste projeto é solucionar este problema centralizando a comunicação interna gerando indicadores reais para soluções e interpretações dos problemas.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um sistema *web* baseado em uma aplicação de rede social visando melhorar e otimizar a comunicação interna das organizações.

1.2.2 Objetivos Específicos

Com a utilização da aplicação que será desenvolvida e o seu desenvolvimento será possível:

- Registrar informações e preferências de um colaborador.
- Centralizar eventos e reuniões que acontecem na empresa, sabendo quem participou.
- Realizar análises de preferências de grupos ou individuais.
- Centralizar as formas de comunicação interna da empresa em apenas uma aplicação.

1.3 JUSTIFICATIVA

Este trabalho está fundamentado na percepção da dificuldade que as organizações possuem na comunicação interna e na necessidade que a informação seja destinada às pessoas certas e que ela seja adequadamente distribuída. Nas empresas, em geral, para cada

funcionalidade, como, *chat*, *e-mail*, troca e compartilhamento de arquivos e agendamento de reuniões ou eventos internos é utilizado um sistema diferente, sistema de terceiros ou até mesmo desenvolvidos internamente na empresa, consumindo recursos e mão de obra. Um sistema único de comunicação para a empresa e em um formato de aplicação que é familiar aos usuários, como ocorre com as redes sociais, pode atender as necessidades de comunicação interna e distribuição adequada da informação.

Para desenvolvimento da aplicação foram utilizadas tecnologias consideradas recentes e com recursos para criação de aplicações *web*. Esse também é um foco do trabalho, considerando um desafio construir uma aplicação que utilize conceitos e tecnologias que não fazem parte do padrão de desenvolvimento *web*.

1.4 ESTRUTURA DO TRABALHO

No primeiro Capítulo apresenta as considerações iniciais, o objetivo e a justificativa do trabalho. O Capítulo 2 é referente ao referencial teórico que se apresenta o um pouco do universo que a aplicação vai abranger. O Capítulo 3 apresenta os materiais e o método utilizado para a modelagem e implementação da aplicação. Os resultados deste trabalho são apresentados no Capítulo 4 e por fim está a conclusão seguida das referências utilizadas no texto.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta alguns conceitos de rede social e rede social corporativa sendo discorrido sobre seus benefícios na aplicação dela dentro de organizações. Esses conceitos visam um melhor entendimento sobre o assunto, levantando, assim, os requisitos necessários para o desenvolvimento de uma aplicação de rede social corporativa.

2.1 REDES

O interesse no termo “rede” no meio organizacional é recente, porém, o termo não é recente e abrange cada vez mais itens que atendem essa nomenclatura. O sentido inicial ou primitivo da palavra é de uma armadilha para captura de caça ou pesca, referenciando o objeto rede. Esse objeto as pessoas conhecem e sabem identificar que é, basicamente, um conjunto de linhas entrelaçadas com nós formados pelas intersecções das linhas. O conceito foi evoluindo em sua conjuntura já no século XIX, passando a ser usado um termo mais abstrato, assim sendo um conjunto de pontos com mútua comunicação. Segundo Castells (1999) e Fombrun (1997), rede é com um conjunto de nós interconectados. A amplitude desse conceito permite a utilização em outras áreas de conhecimento e simplificando a utilização do conceito “Por transposição, a rede é assim um instrumento de captura de informações” (FANCHINELLI; MARCON; MOINET, 2004).

De maneira mais específica, Tomaél (2005, p.94) conceitualiza a rede como uma estrutura não-linear, descentralizada, flexível, dinâmica, sem limites definidos e auto organizável, estabelece-se por relações horizontais de cooperação. Neste contexto identifica-se a importância da interação entre os atores, considerando que esses atores são “nós” da rede, a intensidade da interação entre os atores promove uma redução no espaço e tempo de entre um ator e outro, com isso, intensifica a comunicação e deixa as informações mais próximas.

2.2 REDES SOCIAIS

A existência de redes sociais não é recente, antes mesmo da Internet as pessoas já eram conectadas por meio de outras formas de comunicação, como correio, telefone. Até os meios de transporte, rotas aéreas e estradas ligam as pessoas de uma certa forma. Com os avanços da Internet as redes ganham mais foco e importância no meio digital.

Segundo Recuero (2009, p.26) as redes sociais na Internet ampliam as possibilidades de conexões e capacidade que elas tinham, assim, elas permitem que ligações do mundo *off-line*, antes não abrangidas, possam ser visualizadas.

Para Marteleto (2001, p.72), as redes sociais representam “[...] um conjunto de participantes autônomos, unindo ideias e recursos em torno de valores e interesses compartilhados”. O ser humano está inserido na sociedade por meio das relações comuns que possuem, tais como família, trabalho, amigos, relações de amizade e geralmente optam por relacionar-se com pessoas que possuem os mesmos interesses pessoais, gostos musicais, filmes, livros, etc. Com isso conseguem saber mais sobre a personalidade e as preferências do indivíduo, validando as preferências de seu círculo social.

De acordo com a análise de Recuero (2009, p.26), identifica-se que o Facebook, Twitter, Instagram, entre outras que não são redes sociais propriamente ditas, são apenas ferramentas ou suporte para as interações que compõem uma rede social. Essas aplicações proporcionam conexões para as pessoas, mas são as pessoas que controlam as redes, elas apenas expressam e influenciam a rede.

Alguns pontos que tiveram grande importância para o crescimento e abrangência de aplicações como o Facebook e também de seus antecessores, foi a de um novo paradigma na *web*. Antes existia a Web 1.0, chamada de primeira geração, caracterizada por ser um ambiente com uma grande quantidade de informação. Nesse contexto, o usuário passou a ter acesso a grande quantidade de conteúdo, porém não era possível alterá-lo e muito menos publicar seus conhecimentos e opiniões, com o surgimento de novas tecnologias, surge então a “Era do Usuário”, assim chamada por Santarosa et al. (2013) de geração interativa, os autores descrevem:

Geração Interativa, produzida sob o conceito da inteligência coletiva e explicitada pelas

múltiplas possibilidades de partilha e cooperação. Os atuais sistemas *web* projetam a personalização da navegação na Web, com programas que percebem especificidades e preferências do usuário, otimizando a capacidade de organizar e analisar informação (SANTAROSA et al. 2013, p. 2).

Morais et al. (2013) discorrem sobre o cenário da Web 2.0:

A Web 2.0 evoluiu para uma plataforma com potencialidades quer para a apresentação de informação, quer para a participação na criação de informação, podendo-se considerar como um conjunto de tecnologias e aplicações de software que permitem interagir, colaborar, criar e partilhar informações (MORAIS et al. 2013, p. 54).

As redes sociais em ambientes digitais possuem as características de acordo com os conceitos de Web 2.0, dentre elas as principais: a combinação de diversas tecnologias associadas à facilidade de uso, mudança de foco da publicação para a participação nas relações, produção e disseminação da informação. A facilidade na produção e disseminação das informações impulsiona o crescimento das redes sociais, o que gera um grande volume de dados, que pode ser explorado para diversos fins, tais como direcionamento de propagandas, audiência de programas de televisão, popularidade de produtos, entre outros.

2.3 BIG DATA NAS REDES SOCIAIS

A análise de dados vem sendo moldada pela disponibilidade, velocidade e volume com que eles são criados. Diariamente, os usuários beneficiam-se da função de autocompletar do Google, que é um exemplo de uma ferramenta de análise e coleta de dados, o Big Data. Por mais que o algoritmo não possa prever o que será pesquisado, ele usa os termos de busca para aprender o que as pessoas estão mais interessadas em saber. Dados para análise não faltam, cada dia um usuário de redes sociais em geral compartilha vários interesses, como filmes, livros, comida, jogos e até mesmo preferências em relacionamentos amorosos. Com o Big Data é possível medir e conhecer mais sobre cada pessoa, organização e até sobre necessidades globais. Com esse conhecimento é possível tomar decisões com mais precisão.

Ainda não existe uma definição completa e de consenso para Big Data, pois o termo é muito amplo. Segundo Breternitz (2013), pode-se usar o termo para designar um conjunto de tendências tecnológicas que permite uma nova abordagem para tratamento e entendimento de grandes conjuntos de dados para fins de tomada de decisões.

Zikipoulos et al (2012) dizem que Big Data se caracteriza por quatro aspectos: volume, velocidade, variedade e veracidade.

Volume refere-se à quantidade de dados gerados digitalmente que cresce exponencialmente. A humanidade nunca produziu tantos dados. Segundo uma publicação de Martin Traverso no dia 06 de novembro de 2013, o Facebook com seus mais de 1 bilhão de usuários ativos já produziram mais de 300 petabytes de dados, sendo 300 milhões de fotos por dia.

Outro aspecto importante é a velocidade, sendo que a velocidade em que os dados podem ser processados é quase tempo real. Esse fator faz com que possam ser processadas as 300 milhões de fotos por dia do Facebook, gerando informações do tipo: identificar colegas de sala de aula, locais onde a pessoa esteve e, por meio de um mecanismo como o reconhecimento de imagem, saber quais são as pessoas com as quais se relaciona a partir das fotos que possuem juntos.

Juntando mais informações como o que é curtido ou escrito as revelações são grandes, como, por exemplo, menções de empresas no Twitter combinadas com a análise do texto publicado, para saber se a publicação foi positiva ou negativa, podem prever quais empresas terão suas ações em queda ou em alta. Esta análise já está sendo usada para realizar investimentos. Esse tipo de análise considera o aspecto de veracidade que segundo Breternitz (2013, p.107), “está relacionado ao fato de que os dados não são ‘perfeitos’, no sentido de que é preciso considerar o quão bons devem ser os dados para que gerem informações úteis e também os custos para torná-los bons”. Veracidade se relaciona à capacidade de gerar informações úteis e oportunas. Variedade tem a ver com as diversas fontes em que os dados podem ser encontrado. A variedade se encontra e todos os tipos de aplicações que geram alguma forma de dados úteis para este tipo de análise.

Com os dados que um usuário gera no FaceBook é possível saber mais sobre sua personalidade do que as pessoas mais próximas sabem, como um irmão, ou seja, o FaceBook conhece mais da pessoa que os seus próprios familiares. Com isso é possível perceber a

capacidade de gerar receita dessas aplicações, afinal sua fonte de lucro vem da publicidade. Sendo, assim, é um engano achar que é uma ferramenta gratuita. Com informações de preferências de consumos de seus usuários, as marcas ou clientes do FaceBook, conseguem atingir ainda mais seus consumidores em potencial.

2.4 REDES SOCIAIS CORPORATIVAS.

No meio corporativo as redes são identificadas de vários formatos, seja em uma conversa informal com um colega no horário do café, um encontro fora do trabalho, convenções e até em reuniões com a finalidade de decisão de um determinado objetivo.

O compartilhamento de conhecimento, informações e documentos é muito importante no meio corporativo, a informação e o conhecimento não devem caber a uma única pessoa, seja por meio de treinamentos, reuniões ou textos informativos, ou seja, cada integrante da rede tem um forte papel para a as informações sejam entregue a cada receptor. Neste contexto, Tomaél (2005) descreve um exemplo da rede organizacional.

Com base no seu dinamismo, as redes, dentro do ambiente organizacional, funcionam com espaço de compartilhamento de informação e do conhecimento. Espaços que podem ser tanto presenciais quanto virtuais, em que pessoas com o mesmo objetivo trocam experiência, criando bases e gerando informações relevante para o setor em que atuam (TOMAÉL,2005. p. 94)

Dirigida diretamente para os funcionários, a melhoria de canais de comunicação internos tem vários benefícios, tanto para o colaborador como para o gestor de recursos, além de aumentar o desenvolvimento das pessoas e de instigar a possibilidade de novas relações, o fluxo de trabalho passa a se tornar mais organizado. De acordo com Berlo (2003) uma organização de qualquer espécie só é possível por meio da comunicação, assim, é desconsiderada a existência de empresas ou organização sem comunicação interna, apenas com a comunicação mal organizada. A comunicação entre os colaboradores e setores da empresa é o que faz ser uma organização e não os setores e colaboradores isolados e desorganizados.

Daft (2008) discorre sobre a comunicação:

A frequência e a atividade de comunicação aumentam conforme aumenta a variedade da tarefa. Problemas frequentes requerem maior intercâmbio de informação para serem resolvidos e garantir apropriada completude de atividades. A direção da comunicação é habitualmente horizontal em unidades de trabalho não-rotineiro e vertical em unidades de trabalho rotineiro. A forma da comunicação varia com a analisabilidade da tarefa. Quando as tarefas são altamente analisáveis, as formas escritas e estatísticas da comunicação (memorandos, relatórios, regras e procedimentos) são frequentes. Quando as tarefas são menos analisáveis, a informação é habitualmente conduzida face a face, ao telefone ou em reuniões de grupo. (p. 258).as tarefas são menos analisáveis, a informação é habitualmente conduzida face a face, ao telefone ou em reuniões de grupo (Daft 2008, p. 258).

A partir do que foi citado é possível afirmar que uma das funções da comunicação é ser uma ferramenta que permite melhoria em um processo de gerenciamento para a empresa. Além disso, é possível categorizar a comunicação como um processo de desenvolvimento e visualizar que a comunicação possa gerar conhecimento para as pessoas, modificando a estrutura e comportamentos. Para um processo evolutivo dentro das empresas é necessário que se passe a entender e praticar a comunicação nessa perspectiva.

Maximiano (2007, p. 296) destaca: “Da comunicação dependem ainda a coordenação entre unidades de trabalho e a eficácia do processo decisório. Muito mais do que isso, o processo de comunicação é uma extensão da linguagem e, como tal, um componente fundamental da condição humana.”

Para Maximiano (2007) existem duas preocupações em relação à comunicação, sendo elas a comunicação entre as pessoas e o mecanismo de integração nas organizações. Portanto, a comunicação passa a ser vista como determinante para a evolução e a melhoria das relações interpessoais, com o intuito de contribuir para o desenvolvimento do produto, conhecimento do negócio e avaliação de desempenho.

Para Marchiori (2010), a comunicação deve passar a construir significado e ser geradora de novos contextos em todos os relacionamentos organizacionais. É preciso entender o que faz sentido para as pessoas em seus respectivos setores, isso é um dos processos que elevam as relações internas a um nível mais avançado. Para isso é preciso promover a interação social dentro da empresa, não apenas em um ambiente formal, mas deve ocorrer em diferentes realidades.

3 MATERIAIS E MÉTODO

Neste Capítulo são apresentados materiais para a modelagem e o desenvolvimento do software visando atender os objetivos definidos e o método utilizado.

3.1 MATERIAIS

O Quadro 1 apresenta as tecnologias utilizadas para analisar, modelar e desenvolver o sistema.

Ferramenta Tecnologia	Versão	Finalidade
JavaScript	ECMAScript v.6	Linguagem de programação de <i>script</i> padronizada pela ECMAScript.
TypeScript	2.3.2	É uma linguagem para desenvolvimento JavaScript.
Node.js	6.10.1	Plataforma de desenvolvimento de aplicações <i>server-side</i> baseadas em rede utilizando JavaScript e o V8 JavaScript Engine.
Express	4.0	<i>Framework</i> de aplicações <i>web</i> para Node.js, disponibiliza um conjunto de recursos que facilitam o desenvolvimento no <i>framework</i> .
Angular	4.0	<i>Framework</i> JavaScript que permite ampliar o vocabulário <i>HyperText Markup Language</i> (HTML) para aplicações <i>web</i> .
Angular Material	1.1.3	<i>Framework front-end</i> para componentes, baseado no padrão Material Design.
git	2.12.1	Sistema de controle de versão distribuído e de gerenciamento de código fonte.
Visual Studio Code	1.11.2	<i>Integrated Development Environment</i> (IDE) de desenvolvimento.
PostgreSQL	9.6.3	Sistema para Gerenciamento de Banco de Dados (SGBD) Relacional
pgAdmin	1.5	Ferramenta de <i>open source</i> para administração e desenvolvimento em banco de dados na plataforma PostgreSQL

Quadro 1 - Ferramentas e tecnologias

3.1.2 JavaScript

A linguagem JavaScript foi criada em 1995 por Brenda Eich e em 1996 foi implementada no *browser* Netscape 2.0 (SAMY, 2010).

A padronização ECMAScript da linguagem é feita por Ecma International, que é a especificação responsável pela padronização no lado do cliente.

Hoje JavaScript não executa apenas no *browser* e no lado do cliente, com o nascimento do Node.js é possível trabalhar com o JavaScript no lado do servidor. Por isso em 2009 nasce a padronização CommonJS que tem por objetivo padronizar o uso do JavaScript fora do navegador (COMMONJS, 20014).

3.1.3 TypeScript

TypeScript é uma linguagem de desenvolvimento JavaScript para escrita de código fortemente tipado que é compilado para JavaScript puro e pode ser executado em qualquer navegador. O TypeScript é uma linguagem *open-source* que foi criado e é mantido pela Microsoft.

3.1.4 SailsJS

Sails.js é um *framework* desenvolvido em Node.js que utiliza a estrutura *Model-View-Controller* (MVC). Segundo sua documentação, ele é recomendado para a construção de *chats*, *dashboard* em tempo real, jogos *multiplayers* entre outros, mas ele pode ser usado para qualquer aplicação *web*.

3.1.5 Node.js

Node.js surgiu no final de 2009, a ideia de Ryan Dahl para criar este sistema surgiu quando ele percebeu que, ao fazer um *upload* de uma imagem no site Flickr, a barra de progresso deste *upload* precisava fazer uma nova requisição para saber a porcentagem que a imagem já havia sido enviada ao servidor, foi nesse momento que ele percebeu o poder das

requisições *non-blocking thread* (HARRIS, 2015).

Segundo o site oficial, o Node.js é uma plataforma baseada no *runtime JS* do Chrome, para aplicações escaláveis em rede. Node.js é baseado em eventos, seu modelo é *no-blocking I/O*. O que o torna leve e eficiente, ideal para aplicações em tempo real e de grande volume de dados.

Segundo Junior (2014) ao descrever o funcionamento do Node.js:

O framework faz uso da JavaScript Engine V8 do Google. A V8 é a implementação JavaScript utilizada pelo navegador Google Chrome, é extremamente rápida, mas necessitou de algumas modificações para que tenha um melhor funcionamento em outros contextos que não sejam o browser. Para que se tenha *I/O* baseado em eventos e *non-blocking*, o Node.js faz uso de bibliotecas C *libev* e *libeio*, desenvolvidas por Marc Lehmann. O Node.js é composto por diversos módulos: módulos que fazem parte do núcleo da plataforma, chamados *core modules*, e módulos desenvolvidos pela comunidade (JUNIOR, 2014, p. 2).

O Event Loop é uma característica muito importante do Node.js, ele permite que o *framework* tenha domínio das mudanças de tarefas, sendo que em outras tecnologias que gerencia isso é o próprio sistema operacional. O responsável por fazer o Event Loop é a biblioteca *libev* (JUNIOR, 2014). Segundo Pereira (2013) sobre Event Loop:

O Event-Loop é o agente responsável por escutar e emitir eventos no sistema. Na prática ele é um loop infinito que a cada iteração verifica em sua fila de eventos se um determinado evento foi emitido. Quando ocorre, é emitido um evento. Ele o executa e envia para fila de executados. Quando um evento está em execução, nós podemos programar qualquer lógica dentro dele e isso tudo acontece graças ao mecanismo de função *callback* do Javascript (PEREIRA, 2013, p. 3).

3.1.6 Express

Express é um *framework web light-weight* criado em 2009 por TJ Holowaychuk (ALMEIDA, 2015). Pereira (2003) afirma que o Express surgiu para facilitar a construção de aplicações, já que para utilizar a *Application Programming Interface (API) HyperText*

Transfer Protocol (HTTP) nativa do Node.js seria necessário alterar muito código a cada nova alteração do sistema, sendo o Express responsável por facilitar essas mudanças constantes.

Esse autor apresenta algumas características desse *framework*:

- *Model-View-Routes* (MVR);
- MVC;
- Roteamento de *Uniform Resource Locator* (URL) via *callbacks*;
- *Middleware*;
- Interface RESTful;
- Suporte a *File Uploads*;
- Configuração baseado em variáveis de ambiente;
- Suporte a *helpers* dinâmicos;
- *Integração com Template Engines*;
- Integração com *Structured Query Language* (SQL) e *Not Only SQL* (NoSQL);

3.1.7 Angular

Angular é um *framework* mantido pela Google que permite escrever aplicações *web*, *mobile* e *desktop*. Angular é a segunda versão do *framework*, antes Angular.js, porém, não é uma evolução do primeiro, o *framework* foi reescrito contendo melhorias no desempenho e a utilização do TypeScript com linguagem principal de desenvolvimento.

3.1.8 MongoDB

MongoDB é um banco de dados NoSQL mantido pela empresa 10gen, escrito em C/C++. Sua interface de manipulação de dados é feita em JavaScript e a sua persistência é feita com *JavaScript Object Notation* (JSON). É utilizado no banco o conceito de *schema-less*, ou seja, não existe relacionamento de tabelas ou chave primária, mas sim, *documents* com *embedded documents* mantidas dentro de uma coleção. Outra característica do MongoDB é sua inserção e remoção serem executadas em *runtime*, evitando que uma *collection* fique travada (PEREIRA, 2013).

No MongoDB, por não existir relacionamento entre as coleções, é possível persistir um *embedded document* dentro de um *document*, que é o mesmo que um relacionamento entre tabelas. Porém, isso não é feito da mesma maneira que nos bancos relacionais. Nesse caso é inserido o *document* da relação dentro de outro *document*, evitando assim *selects* e *joins* desnecessários, aumentando então a velocidade na seleção dos dados (PEREIRA, 2013). É nesse contexto que Almeida (2015, p. 128) discorre que “nem todas as propriedades Atomicidade, Consistência, Isolamento e Durabilidade (ACID) estão preservadas nos bancos de dados relacionais são implementadas pelo banco”, ou seja, muitas dessas propriedades precisam estar presentes no desenvolvimento da aplicação e não no banco de dados.

3.2 MÉTODO

O desenvolvimento da aplicação foi dividido em quatro etapas. Na fase um, denominada problemática, foram realizadas observações e entrevistas informais com colaboradores e gestores de empresas para identificar os maiores problemas na comunicação interna e na disseminação de conhecimento nas empresas.

Na segunda etapa, denominada de planejamento, foram criados e analisados os requisitos do *software*. A partir dos requisitos foram criados os diagramas de caso de uso e de classe. Com os diagramas de classe criados foi possível gerar a modelagem dos bancos dados para o banco de dados relacional e NoSQL.

A terceira etapa é a de desenvolvimento e implementação do *software*. O desenvolvimento teve como base a metodologia de desenvolvimento Scrum. Assim, as necessidades foram organizadas em tarefas com uma determinada quantidade de horas estimadas para o desenvolvimento de cada uma. Cada Sprint teve o período de uma semana, sendo repetidas até todas as tarefas serem finalizadas.

No fechamento, que ocorre na quarta etapa, foi realizada a conclusão de todos os pontos requisitados na segunda etapa, com isso, foi realizado um teste geral e com todos os testes unitários obtendo sucesso na execução foi obtida a primeira versão do sistema.

4 RESULTADO

Neste Capítulo são apresentados os resultados obtidos com o desenvolvimento do trabalho. Inicialmente serão apresentados o escopo e modelagem do sistema. Então serão apresentadas as principais telas da aplicação *web* e por fim os códigos fontes gerados durante o desenvolvimento da aplicação.

4.1 ESCOPO DO SISTEMA

O *Minimum Viable Product* (MVP), ou Produto Mínimo Viável, da aplicação deve ser desenvolvido inicialmente para a plataforma *Web*. Porém, a API deve ter suporte para que mais plataformas possam conectar e se comunicar com a base de dados, isso é necessário para evitar retrabalho e alterações muito grandes quando surgir a necessidade de ser produzido um aplicativo para dispositivos móveis.

Neste contexto o sistema deve ter a opção de cadastro de empresa ou organização, pois a necessidade do uso de um software deste segmento não é apenas para empresas. Todo tipo de organização ou grupo que desenvolve trabalhos em conjunto tem a necessidade de troca de informações. Ao criar um registro da empresa será automaticamente criado um usuário administrador, que terá a acesso a todas as alterações necessárias.

Funções básicas como a de usuário deve ser vinculada a uma empresa, mediante de aprovação do usuário administrador. Cada usuário possui uma linha do tempo podendo publicar o que for de seu interesse, todos os usuários podem visualizar essas publicações. A aplicação não faz vínculo de amizade entre os usuários. Um usuário pode deixar de seguir outro usuário se achar que suas publicações são desnecessárias. Uma publicação pode ser comentada e avaliada como boa ou ruim por outros usuários, o usuário que realiza a postagem tem a opção de alterar e remover a publicação.

Na linha do tempo do usuário são apresentadas apenas as suas publicações. Para as publicações em geral o sistema disponibiliza uma linha do tempo comum a todos os usuários. Isso não diferencia o local de realizar a postagem, que será na linha do tempo de cada usuário, a diferenciação ocorre pela forma de filtragem. A linha do tempo principal ordena as

publicações de acordo com a interação de outros usuários. A cada comentário ou aprovação negativa ou positiva, a publicação deve subir para o início da lista.

Cada usuário possui a opção de *chat*, sendo ele individual ou em grupo, sendo que cada um pode criar um grupo e convidar outros usuários para participar do grupo.

A opção de criação de eventos é muito importante para agendamento de reuniões, treinamentos ou palestras. Cada reunião deve ser vinculada a um espaço da empresa e estes espaços são cadastrados pelo usuário administrador, com horários de início e fim. O sistema faz o controle para que não haja eventos no mesmo horário e sala.

Além do MVP há o sistema de troca de arquivo. Cada usuário possui um determinado espaço para armazenamento, com a possibilidade de compartilhar arquivos com outros usuários.

4.2 MODELAGEM DO SISTEMA

Com a pesquisa em assuntos relacionadas às redes sociais, *Big Data* e redes sociais corporativas, foi possível realizar o melhor levantamento de requisitos do sistema. Esse levantamento permitiu visualizar qual seria o melhor comportamento e os problemas de comunicação que a aplicação deveria priorizar a solução.

Foi identificado que a análise dos dados gerados por usuários é de grande importância para a melhoria do processo de comunicação interna. Não é possível identificar um problema sem saber a opinião das pessoas envolvidos, no caso os colaboradores da empresa ou participantes de uma organização.

A adoção de tecnologias que podem trazer melhor desempenho e escalabilidade para aplicação também foi um resultado muito importante desse levantamento. Isso porque o software processará uma grande quantidade de dados para que os resultados sejam relevantes para seus usuários.

O processo de desenvolvimento ficou mais claro e mais rápido após o estudo em áreas relacionadas ao projeto.

Os requisitos funcionais do sistema estão representados no Quadro 2.

Identificação	Nome	Descrição
RF001	Cadastro de organização	Como ação inicial do sistema é necessário realizar o cadastro da organização. Neste cadastro é necessário informar: nome, foto/logo, endereço completo (CEP, rua, número, bairro, cidade, estado e país), CNPJ, quantidade de funcionários, ramo atividade, usuário e senha. Esse usuário será o administrador.
RF002	Cadastro de usuário	Para o cadastro de usuário deve ser informado empresa/organização, nome, telefone, endereço completo (CEP, rua, número, bairro, cidade, estado e país), foto perfil, e-mail, usuário e senha.
RF003	<i>Login</i>	O sistema deve possuir autenticação por e-mail e senha.
RF004	<i>Chat</i>	O sistema deve disponibilizar a opção de <i>chat</i> , para comunicação entre os usuários.
RF005	Criação de grupo de <i>chat</i>	Para o cadastro de grupo de <i>chat</i> é necessário o usuário informar o nome e as pessoas/usuários que possuem acesso ao <i>chat</i> em grupo.
RF006	<i>Timeline</i>	A <i>timeline</i> é a visão das publicações que são comuns para todos os usuários.
RF007	Publicação	Cada usuário pode realizar publicações como fotos, vídeos, textos e URLs de outras páginas.
RF008	Perfil do usuário	O perfil vai mostrar os dados do usuário e uma <i>timeline</i> somente com as suas publicações.
RF009	Cadastro de sala	É possível cadastrar um ambiente físico da empresa, com os seguintes dados: nome, capacidade de pessoas, descrição.
RF010	Cadastro de eventos	O usuário pode cadastrar eventos e convidar participantes. Para cadastrar um evento é necessário selecionar um nome, local e horário. Após isso, o sistema deve validar se existe outro evento no mesmo local e horário, caso não exista o usuário pode prosseguir informando uma descrição e selecionando os convidados para o evento.

Quadro 2 - Listagem dos requisitos funcionais do sistema

No Quadro 3 são apresentados os requisitos não funcionais, que são restritos a regras de negócio do sistema.

Identificação	Nome	Descrição
RNF001	Usuário administrador	Ao ser registrada uma nova organização, deve ser gerado um usuário administrador, para controle geral de funções do sistema.
RNF002	Autorização de cadastro de usuário	Ao ser criado um novo usuário, deve ser apresentada uma notificação para o usuário administrador aprovar sua entrada no grupo de usuários da empresa/organização.
RNF003	Oauth	O usuário deve ter a disponibilidade de usuário o Facebook como autenticação ao sistema, facilitando na criação de seu perfil.

RNF004	Validação de e-mail	O sistema deve enviar um <i>email</i> para a caixa de entrada do <i>email</i> que o usuário informou no cadastro para confirmação do <i>email</i> .
--------	---------------------	---

Quadro 3 - Listagem dos requisitos não funcionais do sistema

De acordo com os requisitos citados nos Quadros 2 e 3, foi criado o diagrama de casos de uso, apresentado na Figura 1.

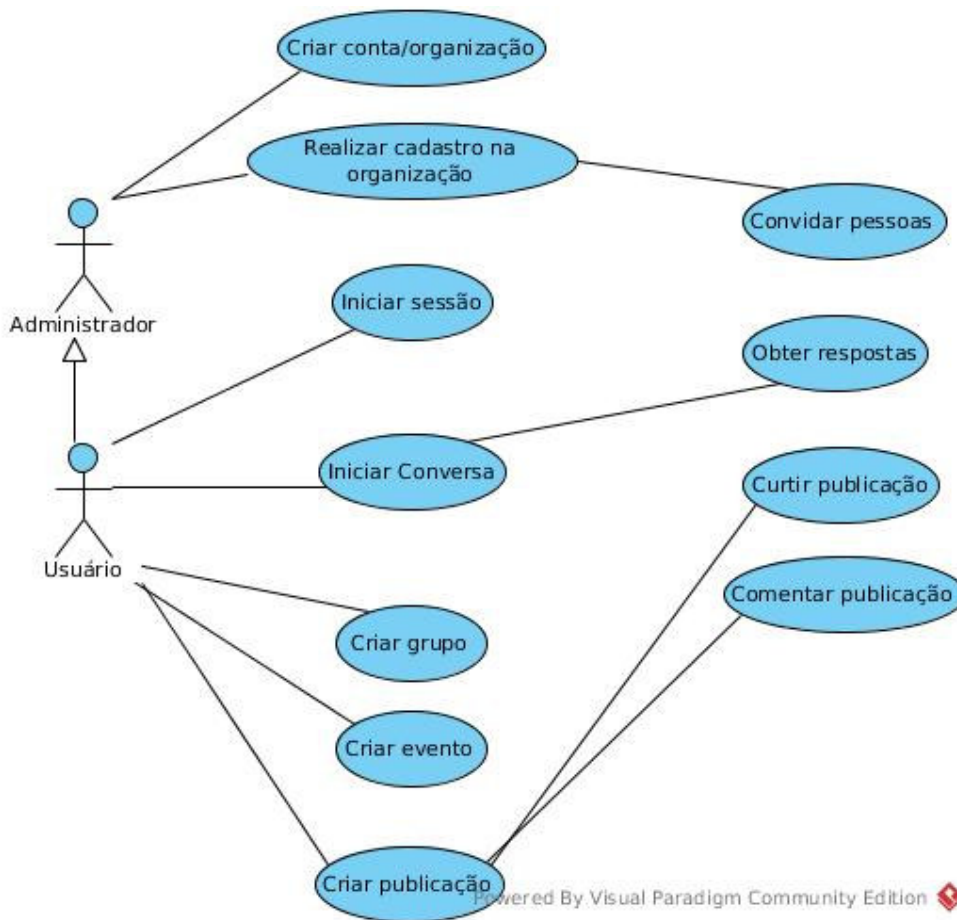


Figura 1 - Diagrama de casos de casos de uso

A partir das necessidades identificadas nos requisitos funcionais e não funcionais, foi criado o diagrama de entidade e relacionamento, o qual pode ser visualizado na Figura 2.

No diagrama de entidade e relacionamento é possível identificar que a aplicação possuirá uma tabela de *account*, a qual será o registro principal, o registro da organização, uma

tabela para um relacionamento inicial. A tabela *user* é o registro de todos os usuários do sistema, que por sua vez tem relacionamento com a tabela *account*, pois todo usuário pertence a uma conta. O usuário também possui relacionamento com as tabelas de endereço, que são representadas por *city*, *state* e *country*, que entre elas cidade possui um estado que possui um país.

A tabela *avatar* representa a imagem de perfil tanto para conta como para o usuário. Cada conta, representada pela tabela *account*, possui um plano, que representa a quantidade de usuário que pode possuir e valor de mensalidade, o plano é representado na tabela *plan*.

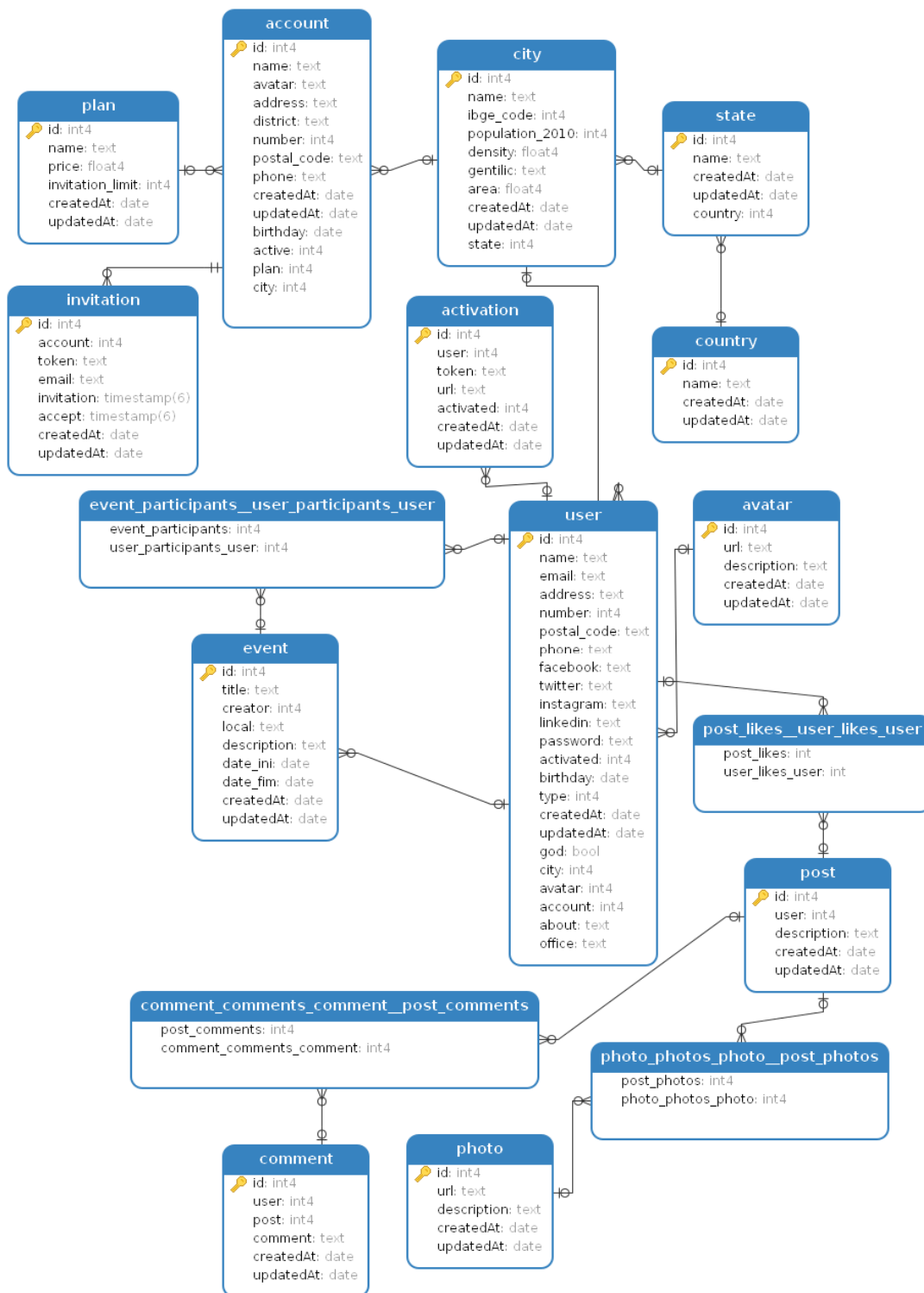


Figura 2 - Diagrama de entidade e relacionamento

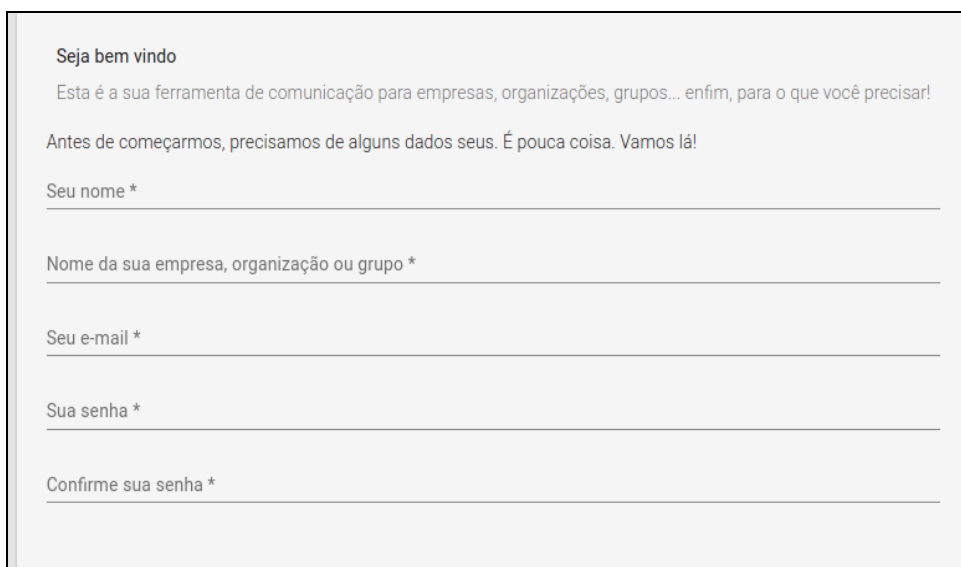
Existe também a possibilidade de criação de grupos, para isso existe a tabela de *group* que relacionará usuários a grupos de acordo com o que for registrado. Foi criada também uma tabela de mensagens, para armazenamento das conversas via *chat* entre usuários e grupos. A tabela de evento armazenará eventos com data, horário, participantes e local do evento, para que seja possível notificar o usuário sobre o evento que foi convidado.

Para que exista a possibilidade de realizar publicações pelo usuário, foi criada a tabela de publicações, que pode armazenar fotos e vídeos. Cada publicação tem uma *timeline* diferente, pois grupos e eventos também possuem uma *timeline* para discussão.

4.3 APRESENTAÇÃO DO SISTEMA

De acordo com os requisitos levantados foram criadas as telas, seguindo o mesmo estilo visual do *Material Design*, criado pela *Google* e já citado neste trabalho.

Para iniciar o uso do sistema é necessário um cadastro inicial de conta. A conta representa a organização que vai utilizar a aplicação, todos os outros registros do sistema, de certa forma são filhos deste cadastro, pois aplicação foi pensada para sustentar várias organizações utilizando a mesma base de dados. Na Figura 3 pode-se visualizar a tela de cadastro desta conta.



Seja bem vindo

Esta é a sua ferramenta de comunicação para empresas, organizações, grupos... enfim, para o que você precisar!

Antes de começarmos, precisamos de alguns dados seus. É pouca coisa. Vamos lá!

Seu nome *

Nome da sua empresa, organização ou grupo *

Seu e-mail *

Sua senha *

Confirme sua senha *

Figura 3 - Cadastro de conta

A partir do cadastro da conta, o sistema cria um registro para a conta e um registro para um usuário administrador. Esse usuário vai gerenciar a conta. Para garantir que o *e-mail* informado é válido, é enviada uma mensagem, que pode ser observada na Figura 4, com um *token* gerado pelo sistema para que o usuário possa acessar a URL anexado ao conteúdo da mensagem. Essa URL redireciona para uma tela inicial, apresentada na Figura 5, com uma mensagem de boas vindas e neste momento é realizada a ativação do registro.



Figura 4 - E-mail de confirmação da conta



Figura 5 - Tela de boas vindas

Após a exibição de uma mensagem de boas vindas e o sistema realizar o processo de ativação da conta, a própria aplicação redireciona o usuário para tela de *login*, como pode ser observado na Figura 6. A partir do *e-mail* e senha cadastrados, o servidor busca o usuário pelo e-mail informado, criptografa a senha passada na requisição, como o mesmo tipo de criptografia que é usado para salvar a senha criptografada no banco. E a partir disso compara as duas senhas, caso sejam iguais é gerado um *token* para que o usuário possa realizar requisições no servidor.

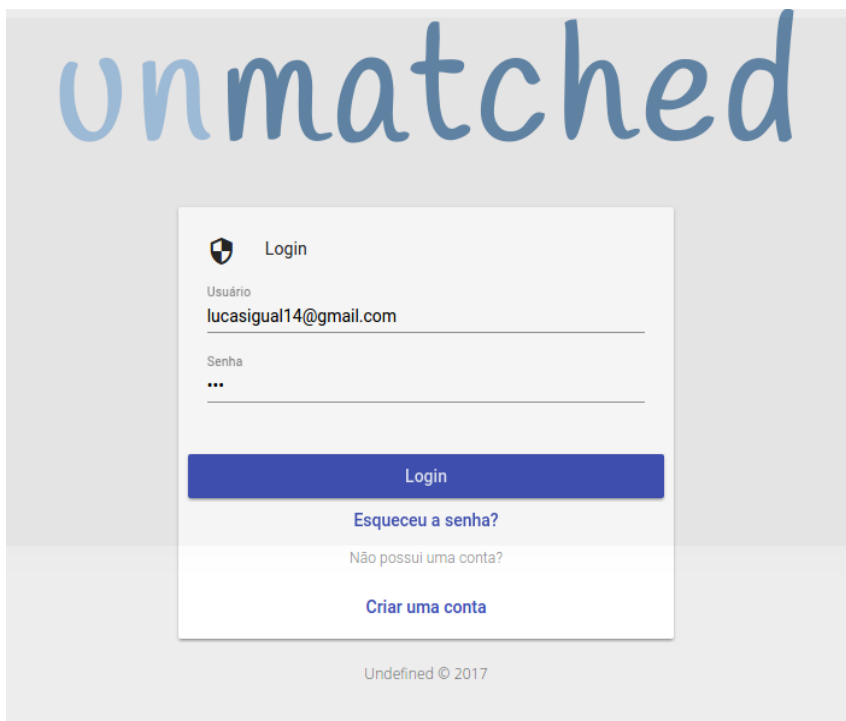


Figura 6 - Tela de login

A visão inicial do sistema é a tela de publicações, chamada de *timeline*, como apresentado na Figura 7, nela são apresentadas todas as publicações realizadas pelos usuários pertencente a mesma organização. Nessa tela o usuário pode curtir uma publicação, comentar e criar publicações, a criação pode conter apenas texto como também imagens e a localização do usuário. Para esse registro não é necessária uma tela nova, mas sim uma janela que se sobrepõe à tela de publicações, janela apresentada na Figura 8.

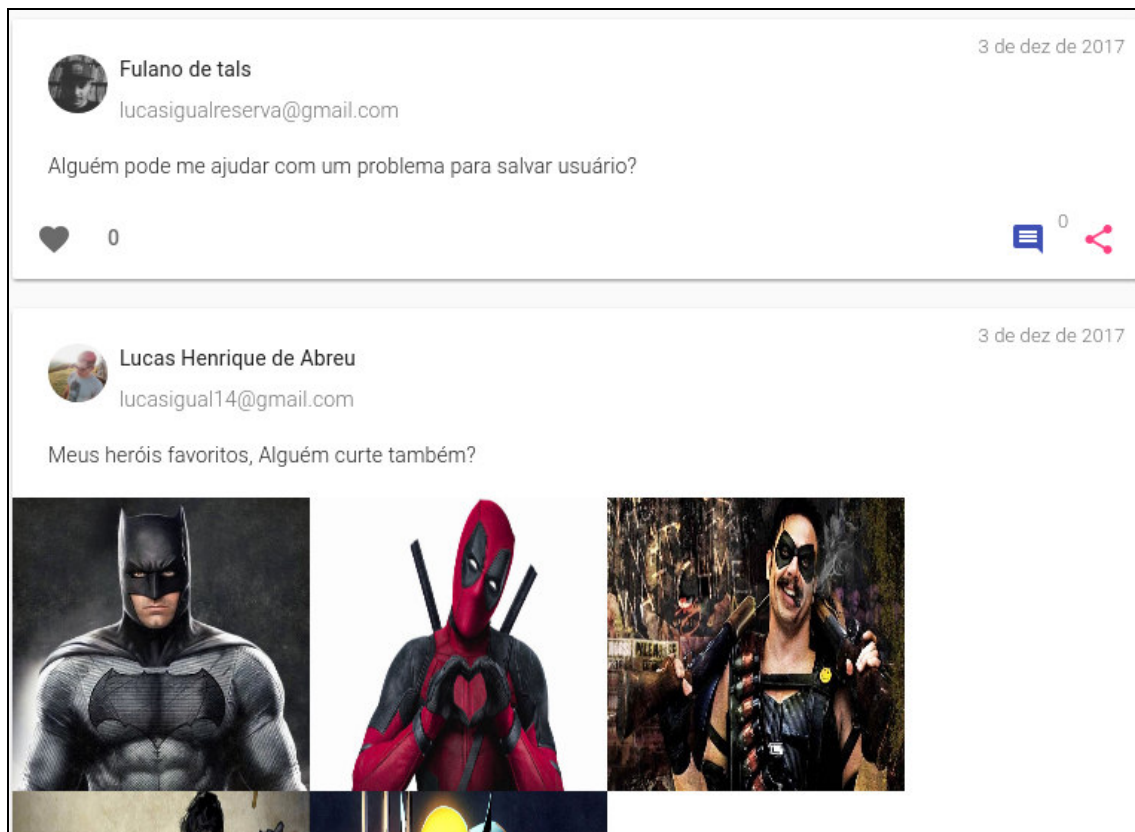


Figura 7 - Tela de postagens

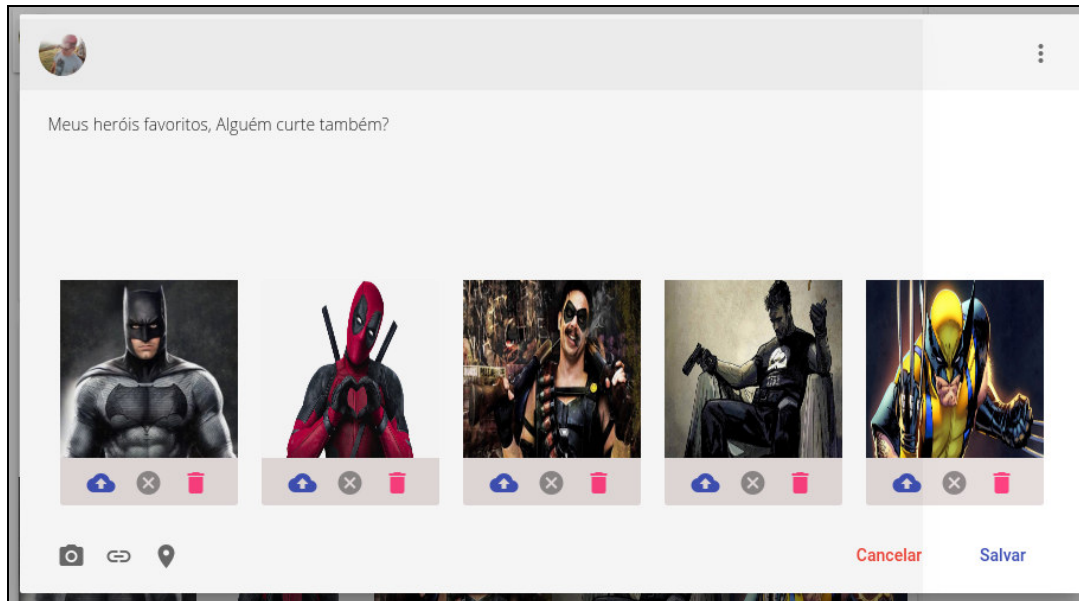


Figura 8 - Janela de criação de postagens

Para que exista interação entre usuários, é necessário ter mais de um usuário vinculado a organização. Para isso foi construída uma área administrativa, na qual o usuário do tipo administrador terá acesso às configurações do sistema. Na primeira versão do sistema, existe apenas a possibilidade de convidar participantes para organização. Para isso basta apenas informar um endereço de *e-mail*, que o convite vai direto para caixa de entrada do convidado, a tela de convites é apresentada nas Figuras 9 e 10.

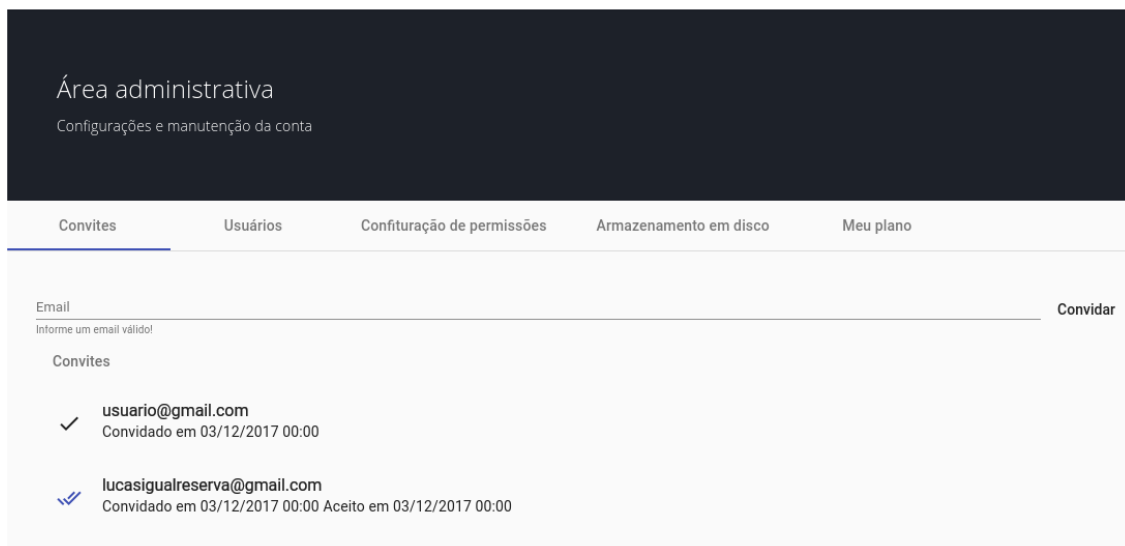
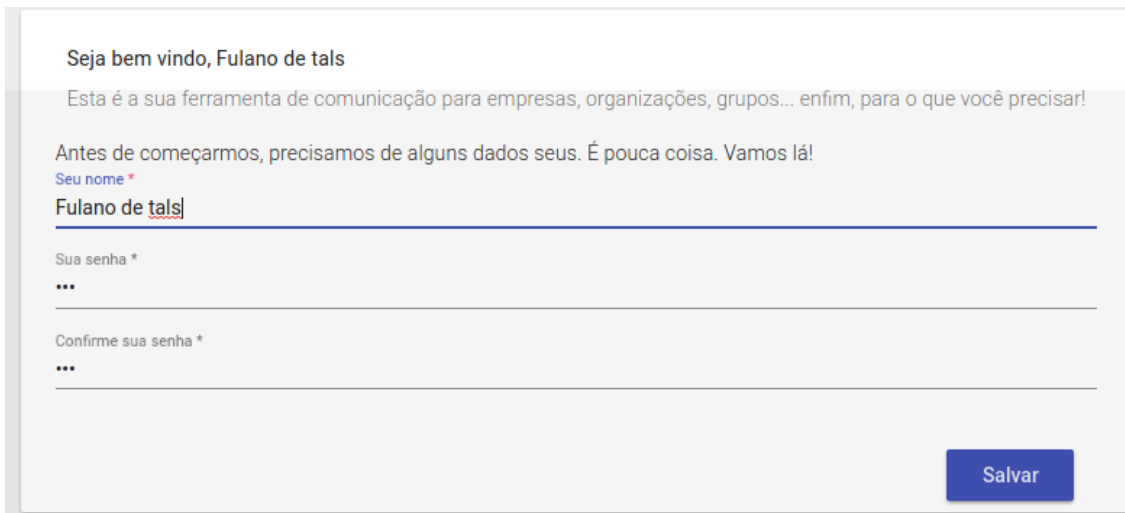


Figura 9 - Aba de convites



Seja bem vindo, Fulano de tals

Esta é a sua ferramenta de comunicação para empresas, organizações, grupos... enfim, para o que você precisar!

Antes de começarmos, precisamos de alguns dados seus. É pouca coisa. Vamos lá!

Seu nome *

Fulano de tals

Sua senha *

...


Confirme sua senha *

...

Salvar

Figura 11 - Pré-cadastro do usuário

Quando o usuário é registrado, o sistema automaticamente realiza a sua autenticação. O usuário possui acesso ao seu perfil, podendo, assim, complementar o cadastro com mais informações e também visualizar suas publicações e eventos que pertencem. Essas telas podem ser observadas nas Figuras 12, 13 e 14.



Publicações Sobre Grupos Eventos

Nome
Fulano de tals

Cargo
Programador

Facebook

Twitter

linkedin

instagram

Cancelar Salvar

Figura 12 - Perfil - dados pessoais

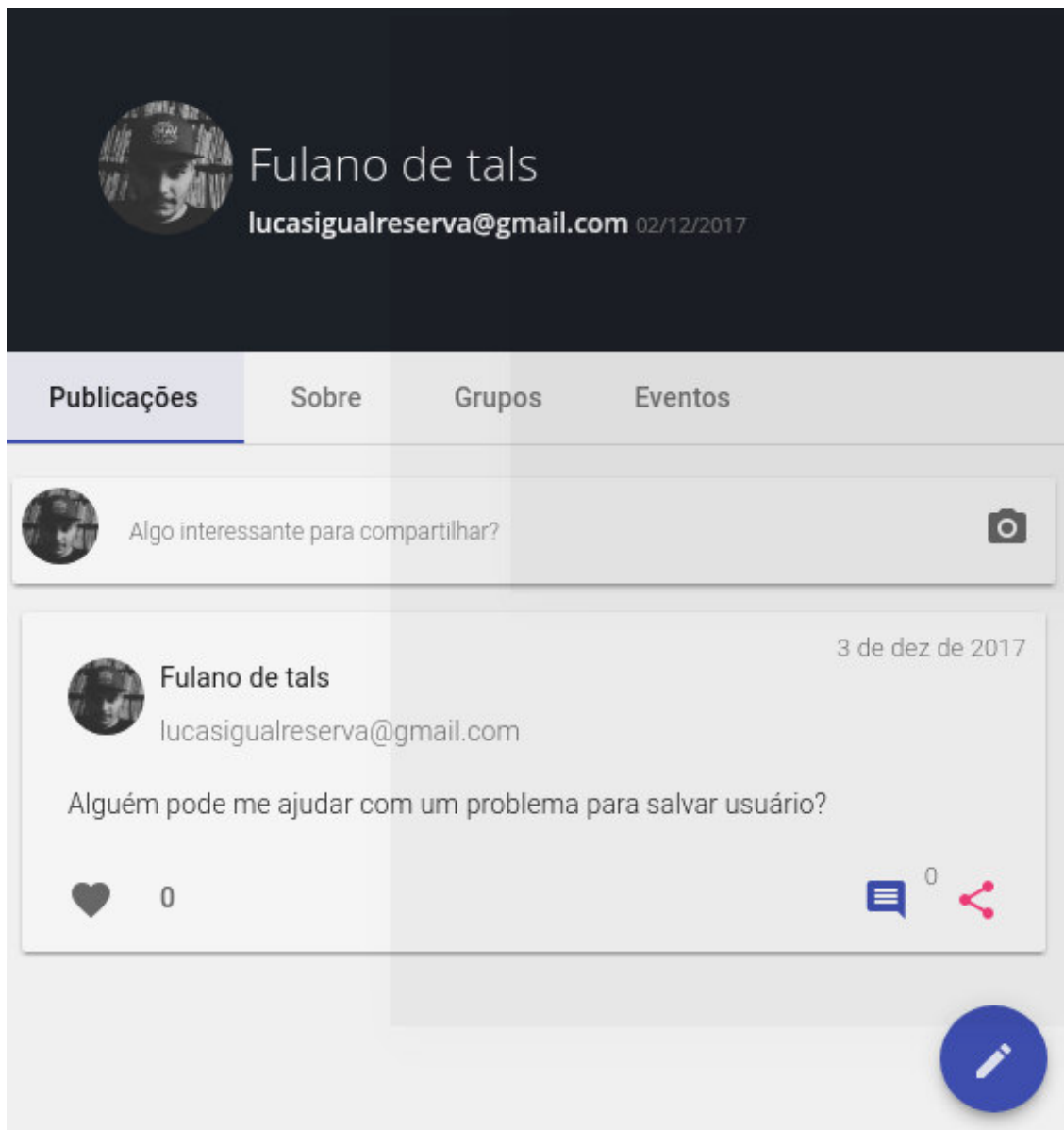


Figura 13 - Perfil - publicações

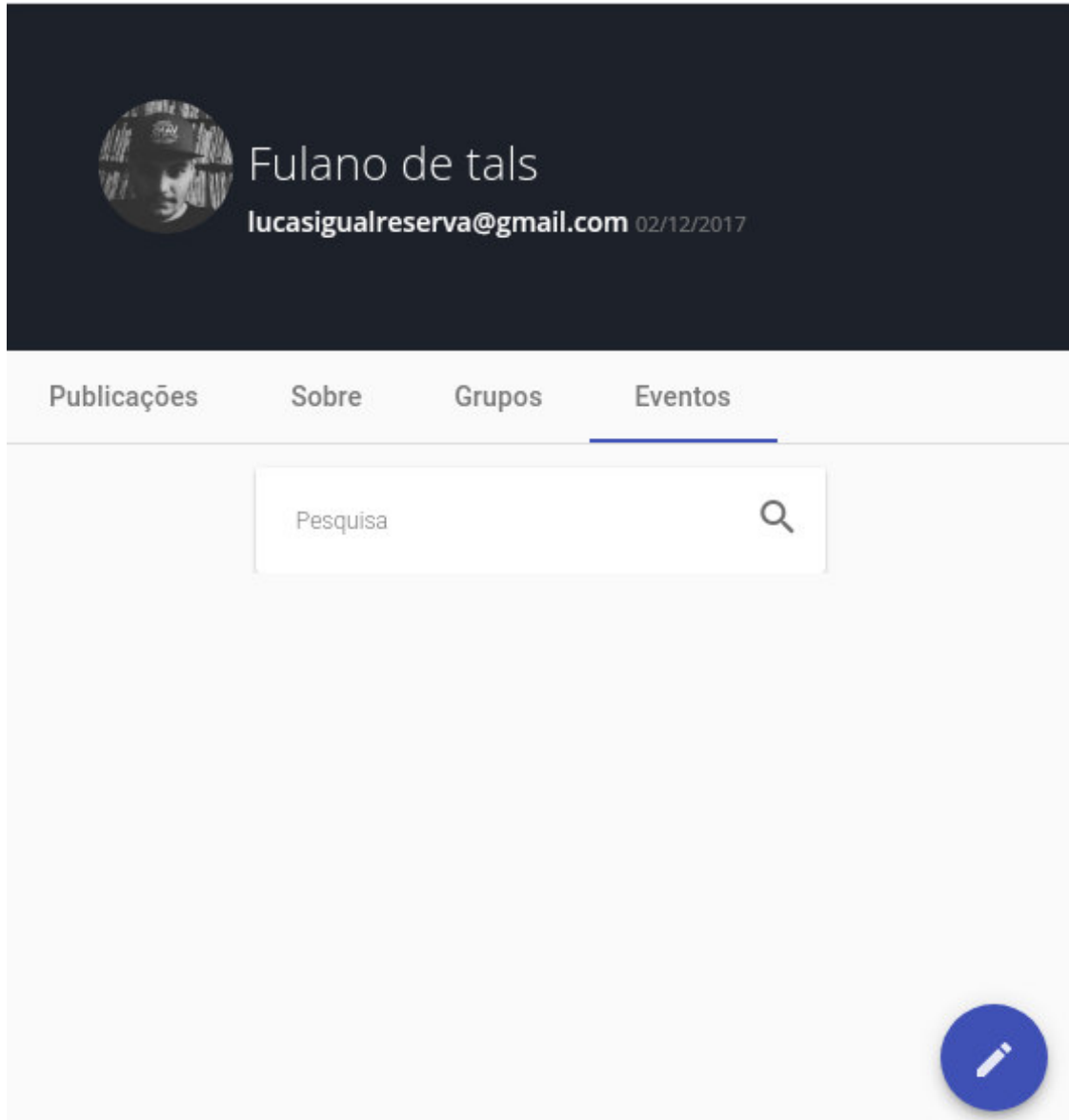


Figura 14 - Perfil - eventos do usuário

Como citado nos requisitos do sistema, usuário pode criar eventos, que são utilizados para notificar outros usuários de reuniões, palestras ou qualquer tipo de evento que a organização esteja promovendo. Para essa função o sistema possui uma tela de eventos, que lista todos os eventos e também com a opção de criar eventos em uma janela que se sobrepõe à tela de eventos, quando o usuário clicar em um evento ele pode visualizar dados do evento. Essas telas são apresentadas nas Figuras 15, 16 e 17.

Pesquisa 

Palestra com professor da UTDPR



Palestra com professor da UTDPR, sobre a qualidade de ensino nas universidades federais.

De 3 de dez de 2017 até 3 de dez de 2017



Figura 15 - Listagem de eventos

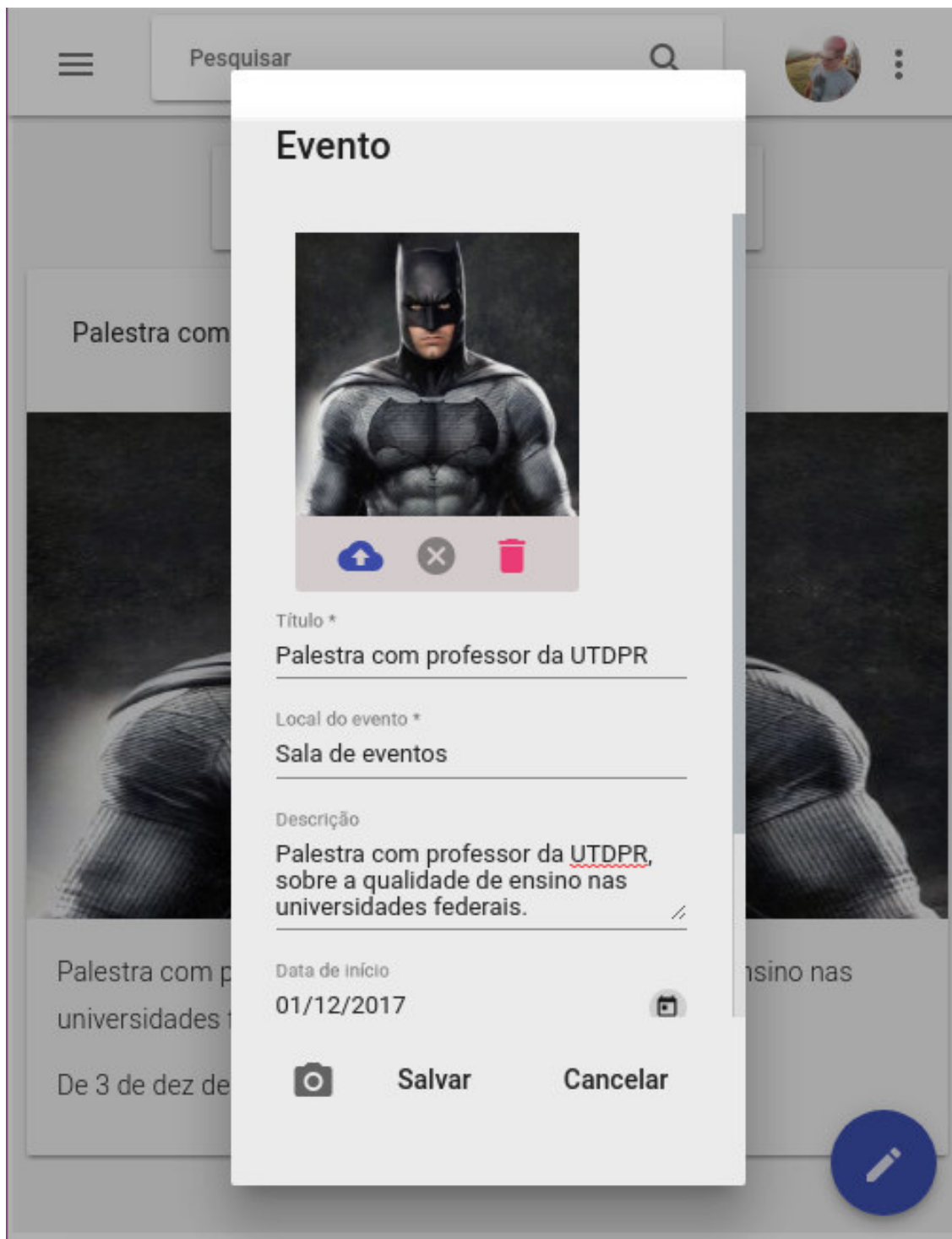


Figura 16 - Criação de eventos

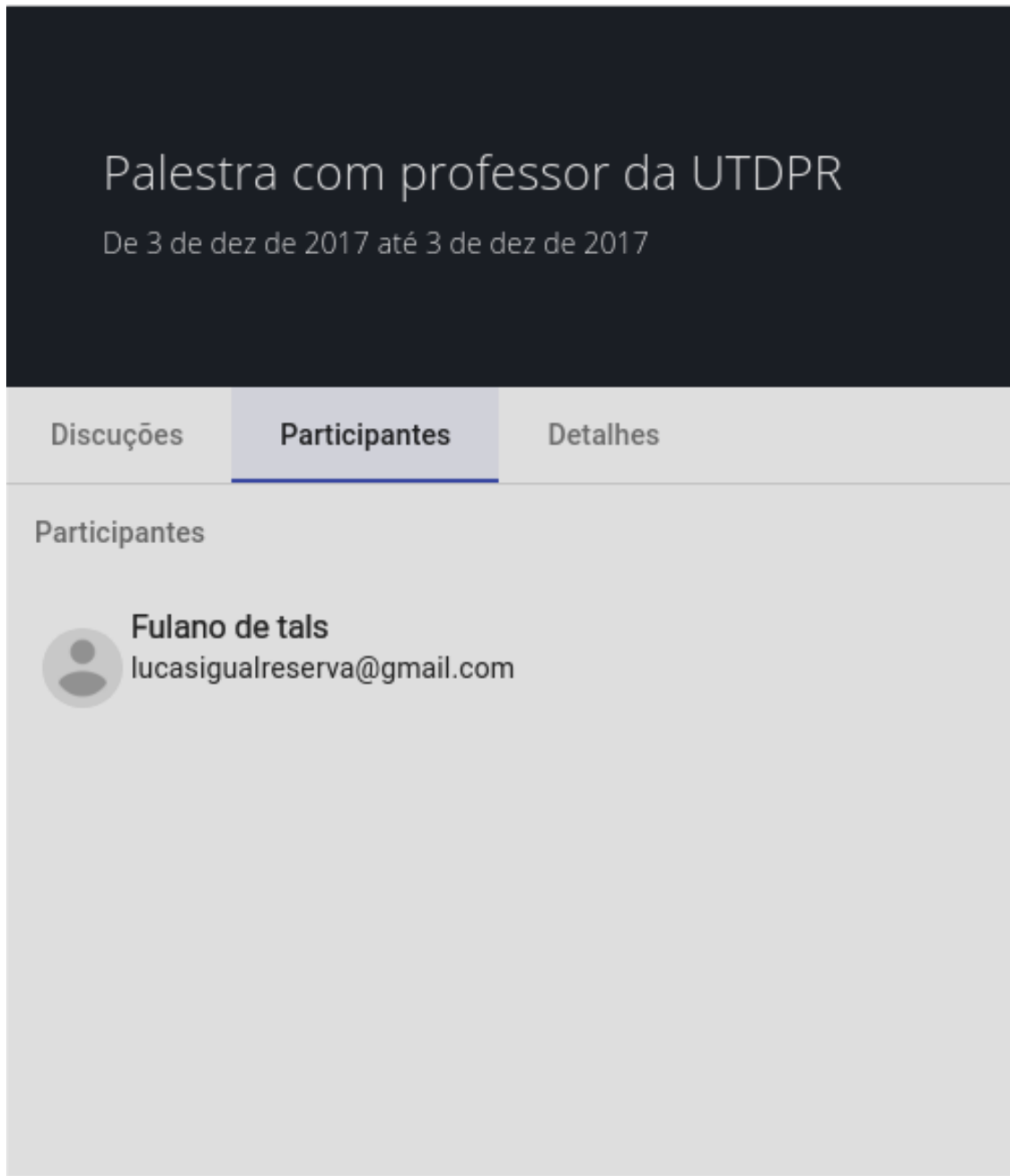


Figura 17 - Visão detalhada do evento

A visão principal da aplicação, contendo os menus de acesso e de navegação superior são exibidas na Figura 18. A parte superior contém os seguintes componentes: botão para abrir e fechar o menu lateral, um campo de pesquisa que ainda não foi implementado, um menu de navegação do usuário que dá acesso ao perfil e sair da aplicação e outro botão que

abre o componente que futuramente vai dar acesso ao *chat* e visualização de atividades. O menu lateral contém os itens de acesso para as páginas que o usuário autenticado tem acesso.

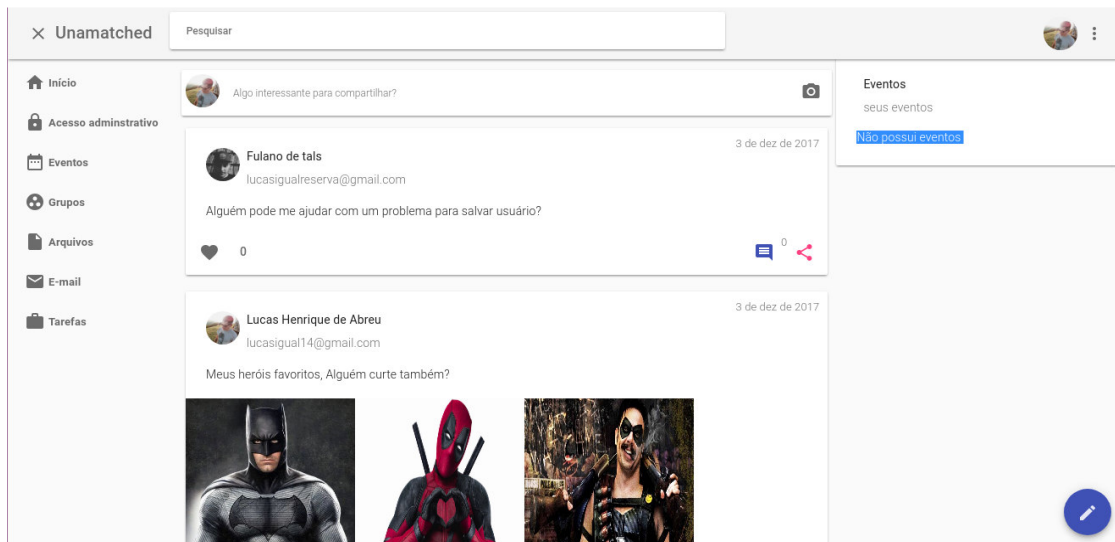


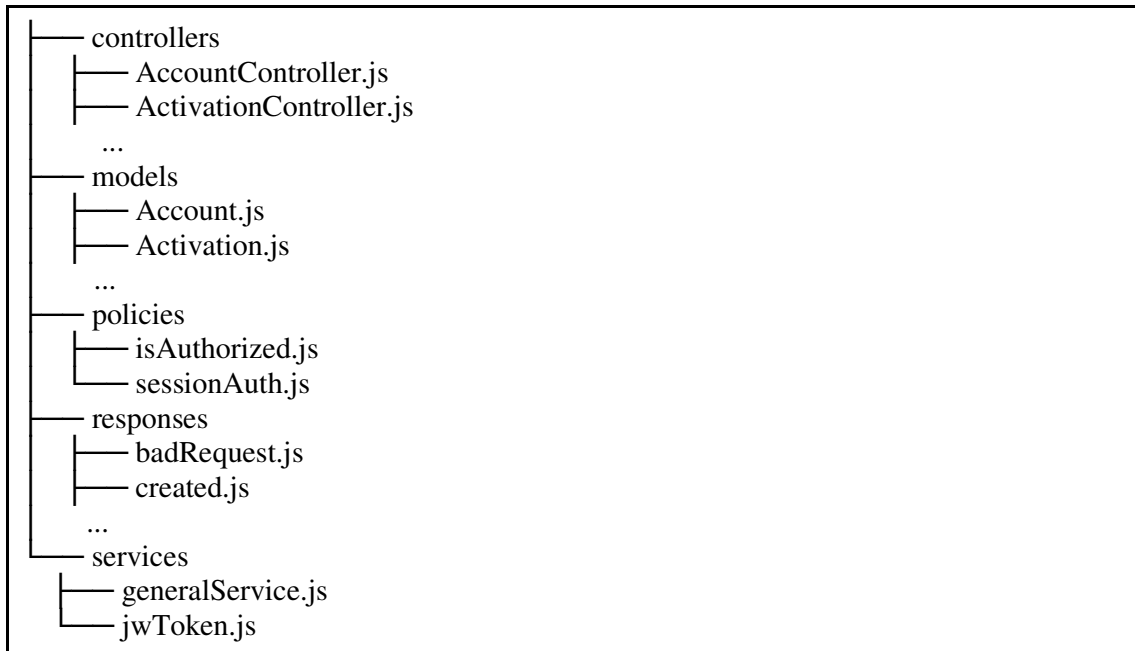
Figura 18 - Visão principal da aplicação

4.4 IMPLEMENTAÇÃO DO SISTEMA

O projeto foi desenvolvido com uma estrutura que separa o lado do servidor do cliente. Cada serviço executará em uma porta e até em máquinas diferentes. Isso foi feito prevendo que no futuro serão desenvolvidas outras aplicações consumindo os dados que a API fornece.

4.4.1 Servidor

No lado do servidor, foi utilizado o *framework SailsJS*. O *SailsJS* é um *framework* de desenvolvimento para Node.js, por isso é necessário um *server* Node.js em execução, essa tarefa o *framework Sails* faz automaticamente. Para iniciar um novo projeto basta executar o comando: `$ sails new nomedoprojeto`. Essa ação gera uma estrutura de diretórios seguindo o padrão MVC, a estrutura da API pode ser observada na Listagem 1.



Listagem 1 - Estrutura de diretórios da API

Segundo a documentação do *SailsJS*¹, os *controllers*, que representam a letra C da sigla MVC, são os principais objetos na aplicação *Sails*, são responsáveis por responder as requisições HTTP, sejam elas de um navegador de internet, aplicativo móvel ou qualquer outro sistema capaz de comunicar-se com um servidor. Eles geralmente agem como intermediários entre seus *models* e *views*. Para muitas aplicações, os controladores conterão a maior parte da lógica do projeto.

Um *model*, que representa a letra M da sigla MVC, é uma coleção de dados estruturados, geralmente correspondendo a uma tabela em um banco de dados. Eles podem ser acessados a partir dos *controllers*, *policieis*, *services*, *responses*, *tests* e em métodos de *models* personalizados. Existem muitos métodos internos disponíveis para cada *model*, entre eles os mais importantes são os métodos: *find*, *create*, *update*, and *destroy*. Esses métodos são assíncronos, e são disponibilizados graças ao *Object Relational Mapper* (ORM) chamado *Waterline* que já implementa essas funções por padrão. Essa funcionalidade disponibilizada pelo *framework*, economiza tempo de trabalho e diminui a quantidade de código a ser escrito. Sem escrever uma linha de código é possível realizar as funções básicas de *create*, *read*,

¹A documentação pode ser encontrada em: <https://sailsjs.com/documentation/concepts/controllers>

update and delete (CRUD) apenas seguindo o padrão *Representational State Transfer* (REST), que para cada ação que se deseja fazer na rota deve ser utilizado um verbo HTTP, como visualizado na Listagem 2.

GET	:	/:controller	=>	findAll()
GET	:	/:controller/read/:id	=>	find(id)
POST	:	/:controller/create	=>	create()
POST	:	/:controller/create/:id	=>	create(id)
PUT	:	/:controller/update/:id	=>	update(id)
DELETE	:	/:controller/destroy/:id	=>	destroy(id)

Listagem 2 - Padrão REST de requisições

O *Waterline* deve enviar uma consulta ao banco de dados e aguardar uma resposta. Consequentemente, os métodos retornam um objeto com o resultado da consulta. Para realmente executar uma consulta, a função *exec(cb)* deve ser chamada neste resultado, em que *cb* é uma função de retorno chamada para executar após a conclusão da consulta.

A *Waterline* também inclui suporte para *promises*. Em vez de chamar *.exec()* em um objeto de consulta, podem ser chamadas as funções *.then()*, *.spread()* ou *.catch()*, retornando uma promessa.

As *Policies* em *Sails* são ferramentas versáteis para autorização e controle de acesso, fornecem à aplicação a disponibilidade de permitir ou negar o acesso aos seus *controllers*. Por exemplo, no caso do projeto desenvolvido neste trabalho, antes de permitir que um usuário faça qualquer requisição do servidor, é usada uma *policy* para verificar se ele está autenticado.

As funções de *policies* são basicamente *middlewares* executadas antes dos *controllers*, podendo ser encadeada como muitos delas da forma que aplicação tiver necessidade.

Os *services* são objetos que possuem métodos que podem ser acessados em qualquer lugar da aplicação. Por exemplo, neste trabalho possui o *service jwtToken*, que possui métodos para gerar e decodificar os *tokens* utilizados para autenticação dos usuários. Os *services* são ótimas opções para reutilizar código em aplicações.

Após a criação do projeto basta executar o comando: *\$sails lift*, assim o *Sails* se encarrega de iniciar um *server Node.js*, executando, por padrão, na porta 1337.

O primeiro item desenvolvido foi a validação de sessão e autenticação de usuário, foi

utilizada a *policy isAuthorized* para validar a sessão. Para configurar quais *policies* os *controllers* tem a necessidade de validar antes de sua execução, é atribuído um *array* de *strings* a um atributo com o mesmo nome do *controller*, ou um asterisco quando é necessário que todos os *controllers* sejam validados pelas *policies*. Sendo que cada posição do *array* é o nome de uma *policy*, que serão executadas em formato de fila, se uma delas falhar, o processo não tem continuidade e retorna uma exceção. Na Listagem 3 pode ser observado que todas os *controllers* serão validados pela *policy isAuthorized*, com exceção do *UserController* nos métodos *create* e *register* e no *AuthController* para todos os seus métodos.

```
/*
 * Policies.js
 * For more information on how policies work, see:
 * http://sailsjs.org/#/documentation/concepts/Policies
 */
module.exports.policies = {
  // Todas as rotas são restritas a policy isAuthorized
  '*': ['isAuthorized'],
  // Exceção da regra
  'UserController': {
    'create': true,
    'register': true
  },
  'AuthController': {
    '*': true
  },
  ...
};
```

Listagem 3 - configuração de *policies*

O módulo que pode ser observado na Listagem 3 recebe a *policy isAuthorized* que deve receber três parâmetros, sendo eles: o *req* abreviação de *request*, *res* que é abreviação de *response* e por último o *next* é uma função de retorno chamada para executar próxima *policy* ou acessar o método do *controller* da rota solicitada. Todas as rotas quando requisitadas recebem os parâmetros de requisição e resposta da solicitação HTTP por padrão, isso é necessário para o recebimento de parâmetros e retorno de resultados para quem fez a

requisição.

Apresentada na Listagem 4, a policy *isAuthorized* vai localizar o *token* no cabeçalho da requisição e utilizando o *service jwtToken*, em seu método *verify* vai validar se é um *token* válido.

```

/** isAuthorized
 * @description :: Policy to check if user is authorized with JSON web
 token*/
module.exports = (req, res, next) => {
  let token;
  if (req.headers && req.headers.authorization) {
    let parts = req.headers.authorization.split(' ');
    if (parts.length == 2) {
      let scheme = parts[0],
          credentials = parts[1];
      if (/^Bearer$/i.test(scheme)) {
        token = credentials;
      }
    } else {
      return res.json(401, { err: 'Formato inválido!' });
    }
  } else if (req.param('token')) {
    token = req.param('token');
    // Nós excluimos o token do param para não mexer com
blueprints
    delete req.query.token;
  } else {
    return res.json(401, { err: 'Token não encontrado!' });
  }
  jwtToken.verify(token, (err, token) => {
    if (err) return res.json(401, { err: 'Token inválido!' });
    req.token = token; // Este é o token descriptado ou a carga
útil que você forneceu
    next();
  });
};

```

Listagem 4 - policy responsável por validar o *token* do usuário

A validação de *token* só pode ocorrer após esse *token* ser gerado e isso ocorre no momento da autenticação do usuário. O *controller AuthController* possui o método responsável pela autenticação do usuário, utilizando o *model User*, o *controller* realiza uma busca na base de dados por *e-mail* com o método *findOne*, pois é um atributo único para cada conta. Após ter o retorno do objeto usuário, o método utiliza a biblioteca *bcrypt*, que também é utilizada para criptografar a senha do usuário antes de ser salvo na base de dados. Essa biblioteca contém vários métodos relacionados à criptografia e um deles é o *compare* que realiza a comparação de um texto não criptografado com um texto já criptografado, assim como pode ser observado na Listagem 5. Caso o retorno do método *compare* for verdadeiro, basta gerar um *token*, que já está implementado no *service jwtToken*, passando o código de identificação ou *primary key* do usuário por parâmetro, para que possa ser solicitado posteriormente. Com o *token* gerado é retornado na resposta da requisição para que possa ser armazenado pelo cliente e enviado em novas requisições.

```
/**
 * AuthController
 * @description :: Server-side logic for managing auths
 * @help       :: See http://links.sailsjs.org/docs/controllers
 */
const bcrypt = require('bcrypt-nodejs');
module.exports = {
  //Realiza a autenticação do usuário.
  index: function (req, res) {
    let email = req.param('email'),
        password = req.param('password');

    if (!email || !password) return res.json(401, {
      err: 'Email e senha são necessários'
    });
    User.findOne({ email: email }).then(user => {
      if (!user) return res.forbidden("Email ou senha inválidos");
      bcrypt.compare(password, user.password,
        (err, match) => {
          if (err) return res.forbidden("Email ou senha inválidos");
          if (match) {
```

```

        return res.json({
            token: jwtToken.issue({ id: user.id })
        });
    }
    return res.forbidden("Email ou senha inválidos");
})
}).catch(err => {
    console.log(err);
    return res.forbidden("Erro ao buscar usuário");
})
}
};

```

Listagem 5 - AuthController

Caso o módulo do *controller* possua métodos diferentes dos herdados do *framework*, nada será sobrescrito, mas sim criado um método novo. Um exemplo disso é a função que busca o usuário autenticado, cujo código é apresentado na Listagem 6. Essa função está contida no *AuthController* e pode ser requisitada passando a rota *auth/getusuariobytoken*, é nela que é obtido o *token* passado no cabeçalho da requisição, então ela decodifica o *token* e obtém a *primary key* do usuário que foi passado por parâmetro no momento de criação do *token*, assim é possível fazer uma busca no banco de dados e retornar o objeto que representa o *model* de usuário.

```

/**
 * Retorna o usuário logado a partir do token
 */
getusuariobytoken: function (req, res) {
    let token, idUser;
    if (req.headers && req.headers.authorization) {
        let parts = req.headers.authorization.split(" ");
        scheme = parts[0], credentials = parts[1];

        if (/^Bearer$/i.test(scheme)) token = credentials;
    }

    idUser = jwtToken.getIdUsuarioByToken(token);
}

```

```

    User.findOne({ id: idUser }).then(user => {
      if (!user) return res.forbidden("Usuário não encontrado!");
      return res.json(user);
    }).catch((err) => {
      return res.json(403, { err: err });
    });
  }
}

```

Listagem 6 - Função que busca o usuário logado

Para representar o modelo do usuário com os mesmos dados da tabela *user* no banco de dados, foi criado o *model User*. O *Model* além de conter os métodos herdados do *framework*, contém a função *beforeCreate* que será chamada antes de salvar o usuário e *beforeUpdate* que será chamada antes de alterar o usuário. Estes métodos são implementados pelo *framework* e herdados no *model*, entretanto é possível sobrescrevê-los. Neste caso os métodos foram sobrescritos para criptografar a senha do usuário antes de salvá-la no banco de dados, como pode ser visto na Listagem 7, utilizando a biblioteca *bcrypt* é gerado um *hash* utilizando a senha e um parâmetro *salt*, gerado pela biblioteca, que define a quantidade de caracteres do *hash*.

```

/**
 * User.js
 *
 * @description :: TODO: You might write a short summary of how this
 model works and what it represents here.
 * @docs        :: http://sailsjs.org/documentation/concepts/models-
 and-orm/models
 */
const bcrypt = require('bcrypt-nodejs');
module.exports = {
  tableName: 'user',
  meta: {schemaName: 'unmatched'},
  connection: 'postgresqlServer',
  autosubscribe: ['destroy', 'update'],
  attributes: {
    id: {
      type: 'integer',

```

```

        autoIncrement: true,
        primaryKey: true
    },
    name: 'string',
    ...
},
// Criptografa a senha antes de criar um Usuário
beforeCreate: function (values, next) {
    bcrypt.genSalt(10, (err, salt) => {
        if (err) return next(err);
        bcrypt.hash(values.password, salt, null,
            (err, hash) => {
                if (err) return next(err);
                values.password = hash;
                next();
            })
    });
}
};

```

Listagem 7 - Model de usuário

Quando for necessário realizar comportamentos diferentes do que o padrão proposto pelo ORM, basta sobrescrever os métodos responsáveis. No *controller* de usuário foi necessário sobrescrever o método *create*, porque é necessário gerar um *token* de autenticação do usuário, para que não haja necessidade de realizar cadastro e autenticar no sistema, economizando uma ação, este método pode ser observado na Listagem 8. O que está sendo feito no método é usar o método *create* do *model User*, o qual é um comportamento padrão do método, porém, quando retornado com sucesso, ou seja, foi criado o registro no banco de dados, o método *create* retorna o objeto criado e, com este objeto é obtida a *primary key* do usuário. Então é utilizada a mesma rotina presente na função de autenticação, que é gerar um *token* e retorná-lo na resposta da requisição para que possa ser armazenado pelo cliente e enviado em novas requisições.

```

/**
 * Cria um novo usuário. Método sobrescrito do Blueprint do Sails,

```

```

* para poder criar um token ao ser criado um novo usuário.
*/
const create = (req, res) => {
  let user = req.body;
  user.activated = 0;
  user.god = 0;
  user.type = 0;
  if (user.password !== user.confirmPassword) {
    return res.forbidden('Senhas não bate, que vergonha!');
  }
  console.log(user);
  User.create(user).then(user => {
    if (user) {
      return res.ok({ token: jwtToken.issue({ id: user.id }) });
    }
  }, err => {
    return res.serverError(err);
  });
};
...

```

Listagem 8 - Função sobrescrita para criar usuário

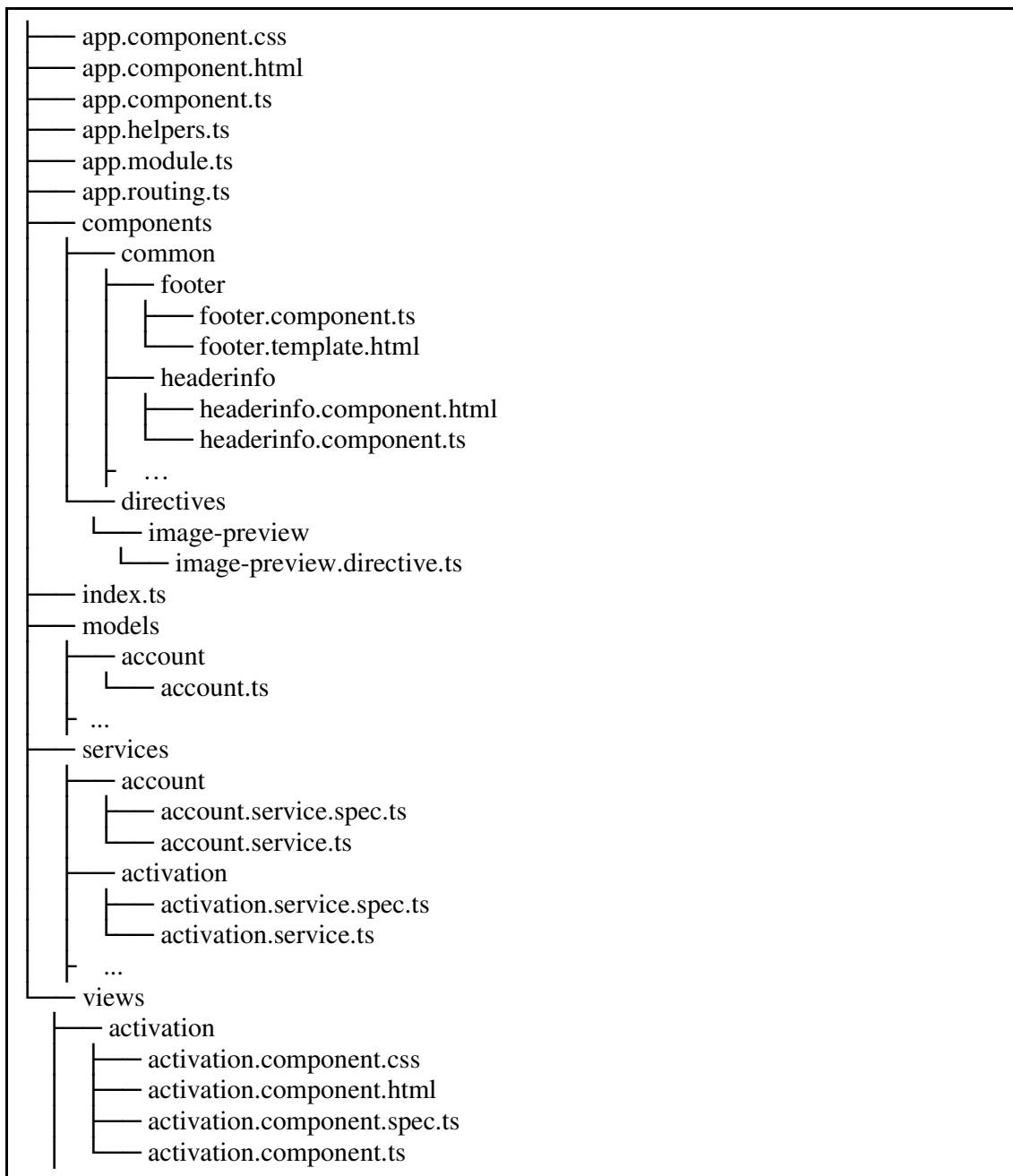
Grande parte das APIs criadas no sistema utilizam o padrão já implementado no *Sails*, porém existem várias outras regras de negócio aplicadas que foram escritas para atender necessidades específicas do sistema, das quais algumas foram citadas nas listagens de código apresentadas neste subcapítulo.

4.4.2 Cliente

No lado do cliente ou *front end* foi utilizado o *framework Angular*. Em suas versões mais recentes, esse *framework* possui o apoio da ferramenta *AngularCli*, que é, basicamente, um apoio ao desenvolvedor. O *AngularCli* gera um projeto inicial, cria componentes no padrão proposto pelo *framework*, possui um *server* para ser utilizado em modo de desenvolvimento, além de realizar o *build* da aplicação para o modo de produção, minificando

os arquivos e deixando os arquivos da aplicação com tamanho físico menor.

Para criar um projeto inicial em *Angular* utilizando o *AngularCli*, basta executar o comando `$ ng new nomedoprojeto` no terminal do sistema operacional. Esse comando cria uma estrutura de diretórios e arquivos para dar início ao desenvolvimento, que pode ser observada na Listagem 9.





Listagem 9 - Estrutura de diretórios do cliente

O arquivo *app.module.ts* contém o módulo principal do projeto. Para declarar uma classe como módulo basta anotá-la como o *decorator* *@NgModule*. De acordo com a documentação do *Angular*², o *NgModules* ajuda a organizar a aplicação em blocos de funcionalidade. O *NgModule* possui um objeto que diz ao *Angular* como compilar e seu código. Ele identifica os próprios *components*, *directives* e *pipes* do módulo, tornando alguns módulos públicos para que os componentes externos possam utilizá-los. O *NgModule* pode adicionar *providers* de *services* aos injetores de dependência da aplicação.

A aplicação possui também um *component* principal, no qual os outros *components* serão renderizados a partir dele. Para declarar uma classe como *component* o *Angular* utiliza o *decorator* *@Component* ele fornece metadados adicionais que determinam como o *component* deve ser processado, instanciado e usado no tempo de execução.

Os *components* são os blocos de construção mais básicos de uma interface em uma aplicação *Angular*. Uma aplicação *Angular* é uma árvore de *components*. Os *components* são subconjuntos de *directives*. Ao contrário das *directives*, os *components* sempre possuem uma *template* HTML e apenas um componente pode ser instanciado por um elemento em um *template*.

Um *component* deve pertencer a um *NgModule* para que ele possa ser usado por outro *component* ou aplicação externa. Para especificar que um componente é um membro de um *NgModule*, ele deve ser listado no campo de declarações do *NgModule*.

Além da configuração de metadados especificada por meio do decorador de *@Component*, os *components* podem controlar seu comportamento em tempo de execução implementando várias interceptações do seu ciclo de vida.

Uma aplicação *Angular* gerencia o que o usuário vê e o que ele pode fazer, isso é feito por meio da interação de uma instância de classe de *component* e seu *template* voltado para o usuário. No *Angular*, o *component* faz a parte do *controller* da *template*, e o *template* representa a página que é exibida para o usuário.

² <https://angular.io/guide/ngmodule>

HTML é a linguagem do *template* Angular. Quase toda a sintaxe HTML é um *template* válido. A tag `<script>` é uma exceção, ela está proibida, eliminando, assim, o risco de ataques de injeção de *script*. Na prática, a tag `<script>` é ignorada e um aviso é apresentado no navegador. É possível ampliar o vocabulário HTML das *templates* com *components* e *directives* que aparecem como novos elementos e atributos.

O *model* não possui um *decorator*, apenas consiste em uma boa prática para o uso do *framework*. O *model* representa um objeto ou entidade, com os atributos que conterà, obrigando que, posteriormente, se possa usar apenas os atributos referentes ao *model* declarado. Para cada objeto do banco, da mesma forma que é criado no servidor, também é criado no cliente.

O *router* no *Angular* permite a navegação de uma interface para a próxima, à medida que os usuários executam esta solicitação. O *router* pode interpretar uma URL do navegador como uma instrução para navegar para uma interface gerada pelo cliente. Ele pode passar parâmetros opcionais ao lado do *component* de exibição de suporte que o ajudam a decidir qual conteúdo específico apresentar. É possível vincular o *router* a *links* em uma página e irá navegar até a visualização apropriada da aplicação quando o usuário clicar nele. E o *router* registra a atividade no diário de histórico do navegador para que os botões de voltar e para frente funcionem.

Para desenvolvimento das interfaces foi utilizado o *framework* de *design* *Angular Material* que possui uma gama de componentes que seguem o padrão de *design* da Google, chamado de *Material desig*. A utilização desse *framework*, também trouxe um ganho de tempo e trabalho, pois a grande maioria dos componentes utilizados nas telas da aplicação são oriundos do *framework*.

A declaração dos *components* internos da aplicação, *providers* de *services* e a importação dos *modules* externos são feitas no módulo principal, o qual é apresentado de maneira resumida na Listagem 10.

```
// App services
import { AccountService } from './services/account/account.service';
...
@NgModule({
  declarations: [
```

```

    AppComponent,
    ...
  ],
  imports: [
    // Angular modules
    BrowserModule,
    ...

    //Material modules
    MatCardModule,
    ...
    RouterModule.forRoot (ROUTES)
  ],
  providers: [
    { provide: LOCALE_ID, useValue: 'pt-BR' },
  ],
})
export class AppModule { }

```

Listagem 10 - exemplo do módulo principal

Para a aplicação ser roteada é necessária uma instância *singleton* do serviço de *router*. Quando a URL do navegador mudar, esse roteador procura uma rota correspondente a partir da qual o *router* pode determinar o *component* a ser exibido. Um *router* não possui rotas até ser configurado. No arquivo *app.routing.ts* que pode ser observado na Listagem 11, são criadas as definições de rota, e é configurado o *router* por meio do método *RouterModule.forRoot* que foi adicionado aos *imports* do *AppModule*, que pode ser visualizado na Listagem 10.

O *array* de rotas do *ROUTES*, que pode ser visualizado na Listagem 11, define como ocorre a navegação entre as visões do sistema. Cada rota mapeia um caminho de URL para um *component*. O *router* analisa e cria a URL, permitindo que se use caminhos relativos e absolutos ao navegar entre as visualizações da aplicação.

No arquivo apresentado na Listagem 11 existem dois pais com seus atributos *path* vazios, apontando para os componentes *basicComponent* e *blankComponent*. Estas duas rotas pai possuem rotas filhas. O *component basicComponent* contém os menus e outros

componentes que apenas usuários logados na aplicação podem ver, já o *component blankComponent* não possui nada dentro dele, apenas renderiza as telas que não necessitam de autenticação.

Para a rota *eventview* foi utilizada a opção de parâmetros, onde o *id* é um *token* para um parâmetro de rota. Em uma URL como */eventview/1*, “1” é o valor do parâmetro *id*. O *ViewEventComponent* correspondente à rota, usará esse valor para encontrar e apresentar o evento cujo *id* for 1.

O “**” caminho na última rota é um curinga. O roteador selecionará essa rota se o URL solicitada não corresponder a nenhum caminho para rotas definidas anteriormente na configuração. Com isso é utilizado o *component PageNotFoundComponent* que exibe uma página “404 - Não encontrado”.

```
export const ROUTES: Routes = [
  // Main redirect
  { path: '', redirectTo: 'maintimeline', pathMatch: 'full' },
  // App views
  {
    path: '', component: basicComponent,
    children: [
      { path: 'maintimeline', component: MainTimeLineComponent },
      ...
    ]
  },
  {
    path: '', component: blankComponent,
    children: [
      { path: 'login', component: LoginComponent },
      { path: 'eventview/:id', component: ViewEventComponent },
      ...
    ]
  },
  // Handle all other routes
  { path: '**', component: PageNotFoundComponent }
];
```

Listagem 11 - Definição das rotas

Para acessar a API é necessário ter uma autenticação no servidor, gerando um *token*. Para que isso ocorra, na tela de *login* o usuário deverá informar seu e-mail e senha, então com essas credenciais é possível enviar uma requisição para o servidor e obter o *token* de autenticação.

Para recuperar esses valores que o usuário informa na interface é necessário criar uma instância de *FormControl* de um *controller* e vincular a um elemento do formulário. A instância do *FormControl* rastreia o valor, a interação do usuário e o *status* de validação do *controller* e mantém a exibição sincronizada com o *controller do component*. Assim, é possível atribuir os valores que o usuário informa nos campos de entrada de dados diretamente aos atributos do *model LoginMode*. Como pode ser visualizado na Figura 12, as variáveis *loginModel.email* e *loginModel.password* conterão o mesmo valor na interface e no *controller*.

```
<form role="form" name="form" #f="ngForm" novalidate>
  <mat-form-field class="full-width">
    <input matInput placeholder="Usuário" [formControl]="emailFormControl"
[(ngModel)]="loginModel.email">
    <mat-error *ngIf="emailFormControl.hasError('required')">
      Usuário é
      <strong>necessário</strong>
    </mat-error>
  </mat-form-field>

  <mat-form-field class="full-width">
    <input matInput type="password" placeholder="Senha"
[formControl]="passwordFormControll" [(ngModel)]="loginModel.password">
    <mat-error *ngIf="passwordFormControll.hasError('required')">
      Senha é
      <strong>necessário</strong>
    </mat-error>
  </mat-form-field>
</form>
```

Listagem 12 - Formulário de login

O *controller do component* de *login* contém apenas o método de autenticação e pode

ser observado na Listagem 13. O método de autenticação realiza apenas a intermediação dos atributos em tela para a chamada do método do *service* de autenticação. Os componentes não devem buscar ou salvar dados diretamente e eles certamente não devem apresentar dados falsos conscientemente. Eles devem concentrar-se na apresentação de dados e delegar o acesso a dados em um serviço. O *AuthService* possui o método *login* que pode ser observado na Listagem 14. Esse método é responsável por enviar a requisição HTTP do tipo *POST* para o servidor, retornando uma *Observable* do tipo *boolean*. No caso de sucesso na autenticação o *token* recebido na resposta da requisição é armazenado na memória local do navegador, com isso o método *login* no *service AuthService* já pode ser responder com um valor *true*. Ao receber essa confirmação o método de *login* no *controller* do *component* muda a rota para o componente de publicações.

```

/**
 * Envia as credenciais para o service de login.
 */
login(loginModel: LoginModel) {
  this.authService.login(loginModel.email, loginModel.password)
    .subscribe(result => {
      if (result) {
        this.router.navigate(['/maintimeline']);
      } else {
        console.log(result);
        this.error = 'Usuário ou senha incorretos';
      }
    }, err => {
      this.error = err.err;
    });
}

```

Listagem 13 - Método de login no modelo de controle

```

login(email: string, senha: string): Observable < boolean > {
  let self = this;
  const params = new URLSearchParams();
  params.append('email', email);

```

```

params.append('password', senha);

const headers = new Headers({
  'Content-type': 'application/x-www-form-urlencoded',
  'Authorization': 'Basic YXBwOmFwcA=='
});
const options = new RequestOptions({ headers: headers });
const url = `${self.apiUrl}auth`;

return self.http.post(url, params.toString(), options).map((response:
Response) => {
  let token = response.json() && response.json().token;
  if (token) {
    self.setToken(token);
    return true;
  } else {
    return false;
  }
});
}

```

Listagem 14 - Método de login no serviço de autenticação

O padrão para as telas do sistema é a realização das ações de CRUD, salvar, alterar, excluir e listar os dados. Para exemplificar esses processos será utilizado o componente *MainTimeLineComponent*, que é a tela de publicações do usuário.

Foi criado um *service* chamado *PostService*, que pode ser observado na Listagem 15. Esse *service* é responsável por realizar as requisições HTTP que serão encaminhadas ao servidor. Esse serviço importa o objeto *enviroment* responsável por obter as configurações e os parâmetros que serão lidos pela aplicação de acordo com o modo de execução, seja ele em produção ou em desenvolvimento. Para esse serviço é utilizado o atribuído *api_url* que contém a URL do servidor e é atribuído para variável *apiUrl*.

O *HttpClient*, utilizado em todos os serviços da aplicação, fornece uma API simplificada para a funcionalidade HTTP para uso com aplicações *Angular*, construindo em cima da interface *XMLHttpRequest* fornecida pelos navegadores. Das vantagens que o

HttpClient fornece, vale citar a forte tipagem de objetos nas requisições e resposta e suporte *interceptor* de requisição. Ele foi utilizado neste projeto para adicionar o *token* do usuário ao cabeçalho da requisição e para melhor gerenciamento de erros via APIs com base em *Observables*.

Cada um dos métodos que o *service* possui retorna um *Observable* do tipo do *model* que deseja retornar. No caso do *PostService* é retornado o *model Post*, seja ele um *objeto* ou uma lista de objetos. Os *Observables* retornam uma *promise* para o método que está realizando a chamada. Sendo uma execução assíncrona, quando a resposta da requisição chegar, é realizada a chamada do método *subscribe* que funciona como uma função de *callback*, retornando o resultado esperado.

O método *findAll* realiza uma requisição com método HTTP do tipo GET, para a URL */post/findPosts*, não receber nenhum parâmetro. Isso porque essa função foi criada no servidor e por padrão busca apenas os *posts* que estão relacionados à conta do usuário autenticado, também possui um sistema de ordenação por data, por número de curtidas e quantidade de comentários.

Para salvar um novo *post* o método *save* do *service* espera um parâmetro do tipo *Post*, e tem como retorno um *Observable* do tipo *Post*. Então é necessário realizar uma requisição HTTP com o método POST, para a URL */post* do servidor, enviando o objeto. Caso tenha sucesso, o servidor retorna o registro salvo, assim o *service* retorna para o *component* que realizou a chamada. Os outros métodos, que são responsáveis por excluir e buscar um registro por *id*, são escritos utilizando padrão.

```
@Injectable()
export class PostService {

  apiUrl = environment.api_url;
  constructor(private httpClient: HttpClient) {
  }
  findAll(): Observable<Post[]> {
    return
    this.httpClient.get<Post[]>(`${this.apiUrl}post/findPosts`);
  }
}
```

```

save(post: Post): Observable<Post> {
  if (post.id) {
    this.httpClient.put<Post>(`${this.apiUrl}post/${post.id}`,
post);
  }
  return this.httpClient.post<Post>(`${this.apiUrl}post`, post);
}
...
}

```

Listagem 15 - Serviço para realizar requisições no servidor

Quem consome o serviço *PostService* é o *component* de publicações, classe chamada de *MainTimeLineComponent*. O componente *MainTimeLineComponent* pode ser observado na Listagem 16. *MainTimeLineComponent* implementa a classe *OnInit* que possui o método *ngOnInit*. Esse método executa após os componentes da tela serem inicializados. No caso deste componente estão sendo carregadas as publicações e é buscando o usuário autenticado por meio dos métodos *getAll* e *getCurrentUser*. Esses métodos se comportam da mesma maneira cada um chama um método de seu respectivo *service*, que realizará as requisições no servidor.

```

...
@Component({
  selector: 'app-main-time-line',
  templateUrl: './main-time-line.component.html',
  styleUrls: ['./main-time-line.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class MainTimeLineComponent implements OnInit {
  ...
  ngOnInit() {
    this.getAll();
    this.getCurrentUser();
    ...
  }
  getCurrentUser() {
    this.userService.getUsuarioLogged().subscribe(user => {

```



```

        this.currentUser = user;
    }, err => {
        this.router.navigate(['/login']);
    });
}

getAll() {
    this.postService.findAll().subscribe(res => {
        this.posts = res;
    }, err => {
        this.openSnackBar('Não foi possível carregar os posts.', 'OK');
    })
}
...
}

```

Listagem 16 - MainTimeLineComponent

Devido às questões de *design* foi criado um *component* interno chamado *DialogPos*. Esse *component* importa outro *component* do *framework* Angular Material, chamado *MatDialog*. Para ser possível a utilização desses componentes externos é necessário importar seus *modules* no *module* principal da aplicação. A importação deve conter a chamada do módulo que está contido na pasta de dependências do projeto e adicionando o *module* no *array* de *imports*, a importação pode ser observada na Listagem 17.

```

import {
    ...
    MatDialogModule,
    ...
} from '@angular/material';
...
imports: [
    //Angular Material modules
    ...
    MatDialogModule,
    ...

```

```
],
```

Listagem 17 - Importação de módulos externos

O *component MatDialog* também possui um *service* em seu *module*. Esse *service* possui métodos responsáveis por abrir e fechar o *dialog*. Para utilizar essas funções é necessário criar uma referência deste *service* no construtor do *component MaintTimeLine*. Essa atribuição pode ser observada na Listagem 18.

```
constructor (
    ...
    private dialog: MatDialog,
    ..
) { }
```

Listagem 18 - Declaração do service do Dialog

Para que seja possível o usuário abrir o *dialog* foi adicionado um evento de *click* em um componente da *interface HTML*. Essa declaração pode ser observada na Listagem 19. O evento de *click* que chama o método *createPost* localizado no *component MainTimeLineComponent* pode ser observado na Listagem 20. Esse método, basicamente, cria uma referência do novo *component* que será criado, a constante *dialogRef* recebe o resultado do método *open*, contido no *service* do *component MatDialog*. Esse método recebe dois parâmetros, um deles é um terceiro *component* chamado *DialogPostComponent* que vai conter o comportamento especificamente do *dialog*, o segundo parâmetro é um objeto com um atributo *data*, o que for atribuído neste atributo pode ser recuperado dentro do *DialogPostComponent*, neste caso está sendo passado o usuário logado, pois essa variável existe apenas no escopo do *component MainTimeLineComponent*.

Após obter a referência do *Dialog* será registrado o evento pelo método *afterClose*. Esse método garante que quando o *Dialog* for fechado, executará o método passado em sua função de *callback*.

```
<mat-card fxFlex class="cardCreatePost" (click)="createPost()">
  ...
</mat-card>
```

Listagem 19 - Evento de click

```
createPost() {
  const dialogRef = this.dialog.open(DialogPostComponent, {
    data: this.currentUser
  });

  dialogRef.afterClosed().subscribe(result => {
    this.save(result);
  });
}
```

Listagem 20 - Método de criação do dialog

O *component DialogPostComponent*, que pode ser observado na Listagem 21, inicia criando um objeto com o tipo do *model Post*. Esse objeto está vinculado como os componentes de entrada de dados na *template* do *component*. Para acessar os dados que foram atribuídos na criação do *dialog*, no componente de *dialog*, foi usada a injeção do *token MAT_DIALOG_DATA*, que atribui para uma variável chamada *data* o valor que lhe foi passado, nesse caso o usuário autenticado.

```

...
@Component({
  selector: 'app-dialog-post',
  templateUrl: './dialog-post.component.html',
  styleUrls: ['./dialog-post.component.css'],
  ...
})
export class DialogPostComponent implements OnInit {
  post: Post = new Post();
  ...
  constructor(
    ...
    @Inject(MAT_DIALOG_DATA) public data: any
  ) {
    ...
  }
}

```

Listagem 21 - DialogPostComponent

No arquivo *dialog-post.component.html* que pode ser visualizado na Listagem 22, que é o *template* do *component DialogPostComponent*, o usuário vai informar os dados para registro de um novo *post*, também pode fazer *upload* de imagens que o *post* vai conter. Para salvar o objeto, o botão salvar possui o atributo *mat-dialog-close* e recebe o objeto *post* por parâmetro. O evento de *click* no botão realiza chamada do evento *afterClose*, o código pode ser observado na Listagem 20, ele vai executar o método *save* do *component DialogPostComponent*.

```

...
<mat-dialog-content>
  <form>
    <div fxLayout="column">
      <textarea      class="inputDescpost"      name="description"
[ (ngModel) ]="post.description"  aria-label="Algo interessante para
compartilhar?"

```

```

        placeholder="Algo interessante para
compartilhar?"></textarea>
    </div>
    ...
</mat-dialog-content>
<mat-dialog-actions fxLayout="row">
    <button class="iconPost" (click)="openFileUpload()"
matTooltip="Carregar foto" mat-icon-button>
        <mat-icon>photo_camera</mat-icon>
    </button>
    ...
    <button mat-button color="primary" [mat-dialog-close]="post"
tabindex="1">Salvar</button>
</mat-dialog-actions>
<input #fileInput type="file" id="imgFile" class="hidden" name="file"
accept="image/*" ng2FileSelect [uploader]="uploader"
multiple>

```

Listagem 22 - Template dialog-post.component.html

Para que o usuário tenha autorização para acessar os recursos do servidor, o sistema sempre deverá enviar o *token* que está armazenado na memória local do navegador, no cabeçalho de cada requisição. Para essa tarefa o *Angular* disponibiliza o recurso *HttpInterceptor* que pertence a API do *HttpClient*, com ela é possível interceptar todas as requisições HTTP. Interceptando as requisições é possível adicionar o *token* ao cabeçalho da mensagem HTTP, sem precisar que isso seja implementado no código toda vez que é necessário criar um método para requisitar um recurso no servidor, o serviço que define esse comportamento é exibido na Listagem 23.

```

intercept(req: HttpRequest < any >, next: HttpHandler): Observable
<HttpEvent<any>> {
    const token = this.authService.getAuthorization();
    const authReq = req.clone({
        headers:
            req.headers
                .set('Authorization', `bearer ${token}`)
                .set('Content-Type', 'application/json')
    });
    return next.handle(authReq);
}

```

```

});
const started = Date.now();
return next
  .handle(authReq)
  .do(event => {
    if (environment.showLogRequestTime) {
      if (event instanceof HttpResponse) {
        const elapsed = Date.now() - started;
        console.log(`>>>> Request for ${req.urlWithParams} took ${elapsed}
ms.`);
      }
    }
  }, err => {
    if (err.status === 401) {
      console.log(`>>>> ${err.status}`);
      this.authService.logoutAndToHome();
    }
  });
}

```

Listagem 23 - Serviço de interceptação de requisições http

O fluxo apresentado no componente *MainTimeLineComponent* é basicamente o mesmo utilizado nas demais telas do sistema. Nesse subcapítulo foram demonstradas as funcionalidades mais importantes para uma explicação do comportamento das telas a nível de *frontend*.

5 CONCLUSÃO

Para atingir os objetivos foi necessário obter um amplo conhecimento na área de comunicação e redes sociais, tais conhecimentos foram muito úteis para o levantamento de requisitos do sistema. E, ainda, auxiliou a entender e definir como o sistema deveria se comportar para auxiliar da melhor forma o usuário em um processo de comunicação interna de empresa ou organização.

Além de pesquisas relacionadas à comunicação interna de empresa, foi também pesquisado sobre *BigData* para um melhor entendimento de o que fazer e como utilizar as informações geradas por usuários desse sistema. O principal benefício que as organizações podem ter com o uso de uma plataforma proposta neste trabalho, são os resultados que poderão ser retirados a partir das informações geradas pelos usuários. Isso pode ser grande importância para o processo e contexto da organização pública, privada ou grupo de pessoas que possam beneficiar-se das funcionalidades da aplicação desenvolvida.

O desenvolvimento da aplicação foi iniciando pelo servidor seguindo o padrão de uma *API REST*, com a criação de conta (organização), usuário, convite de usuário, autenticação, criação de eventos e publicações públicas, juntamente com uma aplicação cliente que consome e fornece informações para o servidor.

Foram utilizadas tecnologias para que a aplicação possa funcionar de maneira ágil e seja de fácil utilização para os usuários. As tecnologias foram escolhidas por serem tecnologias inovadoras em termos de recursos e metodologia no meio de desenvolvimento de aplicações web. Essas tecnologias não proporcionam maior escalabilidade e velocidade no processo de desenvolvimento da aplicação.

A maior dificuldade encontrada no desenvolvimento deste projeto foi a pesquisa relacionada à comunicação interna de empresas. É um assunto que parece ser simples, mas não foram encontradas referências significativas que simplifiquem o seu entendimento.

O projeto tem continuidade, os próximos trabalhos previstos são a implementação do algoritmo de ordenação das publicações de acordo com preferências e interações dos usuários. A ideia é que no futuro os usuários possam gerenciar contas de *e-mail* dentro da aplicação, realizar *upload* e *download* de arquivos, podendo compartilhar com outros usuários.

REFERÊNCIAS

ALMEIDA, Flávio. **Mean**: full stack JavaScript para aplicações web com MongoDB, Express, Angular e Node. São Paulo: Casa do Código, 2015.

ALVES, Fábila Santos. **Um estudo das startups no Brasil**. Salvador: UNIVERSIDADE FEDERAL DA BAHIA, 2013.

BAHIA, Benedito Juarez. **Introdução à comunicação empresarial**. Rio de Janeiro: Mauad, 1995.

BERLO, David K. **O processo de comunicação: introdução à teoria e à prática**. São Paulo: M. Fontes, 2003.

BRETERNITZ, Vivaldo José; SILVA, Leandro Augusto. **Big Data**: um novo conceito gerando oportunidades e desafios. São Paulo: RETC, 2013.

CASTELLS, Manuel. **A era da informação: economia, sociedade e cultura**. São Paulo: Paz e Terra, 1999.

DAFT, Richard L. **Organizações: teorias e projetos**. São Paulo: Cengage Learning, 2008.

FOMBRUN, Charles J. **Strategies for network research in organizations**. Academy of Management Review, v.7, p. 280-291, 1997.

HARRIS, Amber. **The birth of Node**: where did it come from? Creator Ryan Dahl Shares the History. Disponível em: <<http://siliconangle.com/blog/2013/04/01/the-birth-of-node-where-did-it-come-from-creator-ryan-dahl-shares-the-history/>> Acesso em: 10 de junho de 2015.

JUNIOR, Francisco de Assis Ribeiro. **Programação orientada a eventos no lado do servidor utilizando Node.js**. Disponível em: <http://www.infobrasil.inf.br/userfiles/16-S3-3-97136-Programa%C3%A7%C3%A3o%20Orientada____.pdf>. Acessado em: 10 de Junho de 2015.

MARCHIORI, Marlene. **Os desafios da comunicação interna nas organizações**. Caxias do Sul: UCS, 2010a. p.156.

MARTELETO, Regina Maria. **Análise de redes sociais: aplicação nos estudos de**

transferência da informação. Ciência da Informação, Brasília, v. 30, n. 1, p. 71-81, jan./abr. 2001.

MAXIMIANO, Antonio C. A. **Introdução à administração.** São Paulo: Atlas, 2007.

PEREIRA, Caio Ribeiro. **Aplicações web realtime com Node.js.** São Paulo: Casa do Código, 2013.

RECUERO, Raquel. **Rede social.** In: SPYER, J. (Org.). Para entender a internet: noções, práticas e desafios da comunicação em rede. São Paulo: NãoZero, 2009. p. 25-26.

SANTAROSA, Lucila Maria Costi; CONFORTO, Debora; SCHNEIDER, Fernanda Chagas. **Tecnologias na web 2.0,** In: Colóquio Luso-Brasileiro de Educação a Distância e Elearning, 3, Lisboa, 2013.

SILVA, Maurício Samy. **JavaScript : guia do programador.** 4 ed. São Paulo: Novatec Editora, 2010.

TOMAÉL, Maria Inês. **Das redes sociais à inovação.** Por o estado. 2005.

ZIKOPOULOS, Paul; DEROOS, Dirk; PARASURAMAN, Krishnan; DEUTSCH, Thomas; CORRIGAN, David; GILES, James. **Harness the power of Big Data: the IBM Big Data Platform.** Emeryville: McGraw-Hill Osborne Media, 2012.