UNIVERSIDADE ESTADUAL DE MATO GROSSO DO SUL
COORDENAÇÃO DO CURSO DE SISTEMAS DE INFORMAÇÃO

ESTUDO COMPARATIVO DAS TECNOLOGIAS DE DESENVOLVIMENTO DE SISTEMAS WEB: POLYMER, DJANGO E METEOR.

Ingrid de Souza Solique

Profa. Dra. Glaucia Gabriel Sass

Prof. Me. André Chastel Lima

Dourados - MS

2018

UNIVERSIDADE ESTADUAL DE MATO GROSSO DO SUL
COORDENAÇÃO DO CURSO DE SISTEMAS DE INFORMAÇÃO

ESTUDO COMPARATIVO DAS TECNOLOGIAS DE DESENVOLVIMENTO DE SISTEMAS WEB: POLYMER, DJANGO E METEOR.

Ingrid de Souza Solique

Outubro de 2018.

Banca Examinadora:

Profa. Dra. Glaucia Gabriel Sass (Orientadora)
Ciência da Computação – UEMS

Prof. Me. André Chastel Lima (Co-Orientador)
Ciência da Computação – UEMS

Prof. Dr. Cleber Valgas Gomes Mira
Sistemas de Informação – UEMS

Profa. Esp. Regiane Marcon
Sistemas de Informação – UEMS

ESTUDO COMPARATIVO DAS TECNOLOGIAS DE DESENVOLVIMENTO DE SISTEMAS WEB: POLYMER, DJANGO E METEOR.

Ingrid de Souza Solique

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Ingrid de Souza Solique e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, Outubro de 2018.

Profa. Dra. Glaucia Gabriel Sass. (Orientadora)
Prof. Me. André Chastel Lima. (Co-Orientador)

AGRADECIMENTOS

Agradeço primeiramente a Deus por todas as oportunidades que tem me dado e por me abençoar em todos os momentos, principalmente naqueles momentos em que não fui merecedora.

Aos meus pais, Edvaldo Dias Solique e Inês Monteiro de Souza Solique, irmãos, Vinícius de Souza Solique e Victor de Souza Solique, e noivo Rafael Carvalho de França que com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

A minha orientadora Profa. Dra. Glaucia Gabriel Sass, e ao meu co-orientador Prof. Me. André Chastel Lima, pelas orientações, correções e incentivos.

A todas as pessoas que direita ou indiretamente me ajudaram, meu muito obrigada.

RESUMO

A Web está em constante evolução e, juntamente com ela, as tecnologias de desenvolvimento de sistemas. Nesse cenário, os desenvolvedores estão sempre buscando acompanhar essa evolução, no entanto materiais que apresentem vantagens e desvantagens sobre essas tecnologias não são facilmente encontrados. Afim de suprir essa lacuna, o presente trabalho propõe um estudo comparativo sobre tecnologias atuais de desenvolvimento de sistemas Web como as tecnologias Polymer, Django e Meteor, bem como sugestões de critérios para comparação dessas. Além disso, iremos desenvolver um roteiro de desenvolvimento de software usando essas tecnologias.

Palavras-chave: Tecnologias de desenvolvimento Web, Estudo Comparativo, Django, Meteor, Polymer.

ABSTRACT

The Web is constantly evolving and along with it the systems development technologies. In this scenario, developers are always on the lookout to follow this evolution, however literature that present advantages and disadvantages about these technologies are not easily found. In order to fill this gap, the present work proposes a comparative study on current Web systems development technologies such as Polymer, Django and Meteor technologies, as well as suggestions for criteria to compare such technologies. Furthermore, we are going to develop a roadmap of software development using these technologies.

Keywords: Web development technologies. Comparative study. Django. Meteor. Polymer.

SUMÁRIO

1 Introdução.....19
1.1 Justificativa 20
1.2 Objetivos.....21
1.2.1 Objetivo geral.....21
1.2.2 Objetivos específicos 21
1.3 Metodologia 22
1.4 Organização dos capítulos.....22
2 Levantamento teórico.....23
2.1 Sistema de informação 23
2.1.1 História dos sistemas de informação 26
2.2 Recursos e conceitos utilizados pelas tecnologias abordadas 28
2.2.1 Banco de dados.....28
2.2.2 HTML 31
2.2.3 CSS.....32
2.2.4 Javascript.....33
2.3 Polymer.....33
2.4 Django 35
2.5 Meteor.....36
3 Resultados.....39
3.1 Polymer.....39
3.1.1 Roteiro de desenvolvimento polymer.....40
3.2 Django 40
3.2.1 Roteiro de desenvolvimento django.....41
3.3 Meteor.....44
3.3.1 Roteiro de desenvolvimento meteor.....44
4 Conclusão.....49
Referências bibliográficas.....51
Apêndice A – Cenário de estudo 55

LISTA DE ABREVIATURAS E SIGLAS

CSS *Cascading Style Sheets* - Folhas de Estilos em Cascata

DOM *Document Object Model* - Modelo de Objeto de Documento

HTML *HyperText Markup Language* - Linguagem para marcação de hipertexto

JSON *JavaScript Object Notation* - Notação de Objetos JavaScript

ORM Object Relational Mapping - Mapeamento Objeto Relacional

SI Sistemas de Informação

SQL *Structured Query Language* - Linguagem de consulta estruturada

XML *eXtensible Markup Language* - Linguagem de marcação extensível

LISTA DE FIGURAS

Figura 1 – Atividades básicas de um Sistema.....24
Figura 2 – Dimensões de um sistema de informação. 25
Figura 3 – Paradigma centralizado.....26
Figura 4 – Paradigma cliente/servidor em 2 camadas 27
Figura 5 – Paradigma cliente/servidor em 3 camadas 28
Figura 6 – Comparação entre o modelo relacional e modelo NoSQL orientado a documentos.....30
Figura 7 - Exemplo de documento NoSQL.....30
Figura 8 – Exemplo de código HTML 32
Figura 9 – Exemplo de código CSS 33
Figura 10 – Página inicial do Blog com Django, apresenta a listagem dos artigos ...56
Figura 11 – Página de visualização dos artigos com Django, apresenta todo o conteúdo do artigo e os comentários – Parte 1.....56
Figura 12 – Página de visualização dos artigos com Django, apresenta todo o conteúdo do artigo e os comentários – Parte 2.....57
Figura 13 – Página inicial do painel administrativo do Django 57
Figura 14 – Página de modificação de artigos com Django – Parte 1.....58
Figura 15 – Página de modificação de artigos com Django – Parte 2.....58
Figura 16 – Página inicial do Blog com Meteor, apresenta a listagem dos artigos e botões para gerenciar os artigos.....59
Figura 17 - Página de visualização dos artigos com Meteor, apresenta todo o conteúdo do artigo e os comentários – Parte 1.....59
Figura 18 - Página de visualização dos artigos com Meteor, apresenta todo o conteúdo do artigo e os comentários – Parte 2.....60
Figura 19 – Página de modificação do artigo com Meteor 60

1 INTRODUÇÃO

A constante evolução da Web, ligada a evolução e criação de novas tecnologias de desenvolvimento Web são as responsáveis por, a todo momento, nos depararmos com novas ferramentas, *frameworks*, bibliotecas e linguagens de programação.

Nesse sentido, a melhoria tanto no produto final apresentado ao usuário, quanto nos processos e ferramentas de desenvolvimento é uma tendência mantida nos últimos anos. Para seguir essa tendência os desenvolvedores devem estar sempre inteirados das tecnologias que estão evoluindo e são inventadas.

Para manter-se sempre atualizados, os desenvolvedores buscam materiais para ajudar a escolher qual a melhor opção de tecnologia para o problema a ser resolvido, ou até mesmo optar por aprender uma ou outra tecnologia. E materiais que apresentem vantagens e desvantagens de tecnologias emergentes ajudam bastante nesse processo de escolha. Exemplos desse tipo de material são:

Mingoia (2014) em seu artigo intitulado “Here’s the difference between Polymer and Angular”, realiza uma análise comparativa entre Polymer e Angular apresentando as diferenças entre eles. Já Strahan (2015) em “MeteorJS vs AngularJS ain't a thing”, compara Meteor e Angular e sugere em qual tipo de cenário é mais interessante usar cada um deles.

Danailov (2016) no artigo “Polymer vs Angular vs React: Using Web Components to Evolve the Experience”, faz uma comparação entre Polymer, Angular e React, baseado em um projeto desenvolvido por sua equipe e apresenta vantagens e desvantagens sobre o Polymer e em alguns momentos descreve como determinada funcionalidade é implementado em Angular e React.

Urdhwareshe (2016) compara React.JS e Polymer em “Polymer vs. React: Comparison between Two Front End Javascript Libraries”. O autor inicia o artigo fazendo uma crítica sobre escolher uma tecnologia baseado puramente na popularidade dela. Conceitua as duas tecnologias e realiza a comparação delas a partir de 10 critérios diferentes, como arquitetura dos componentes, reusabilidade dos componentes, tag HTML personalizadas, catálogo de componentes, entre outros critérios.

Sidorenko (2017) nos apresenta uma lista comparativa de tecnologias emergentes que são tendências em 2017 em “7 Best Frameworks For Web Development in 2017”. Esta lista apresenta vantagens, desvantagens e sugestões de cenários onde a utilização de cada tecnologia teria mais sucesso, entre outras são citadas as seguintes tecnologias: Django, Rudy on Rails, Angular, Meteor, React.

Diante desses estudos, identificou-se a necessidade de realizar um estudo comparativo com as tecnologias Polymer, Django e Meteor, tendo em vista que nas pesquisas realizadas não foram localizados materiais que apresentem comparação entre essas três tecnologias.

Polymer é uma biblioteca JavaScript que tem por objetivo auxiliar os desenvolvedores na criação de elementos HTML reutilizáveis, segundo os princípios de *Web Components* (POLYMER PROJECT B, 2017).

Django é um *framework* Web de alto nível da linguagem de programação Python, que tem como missão abstrair os problemas ligados ao desenvolvimento Web, para que o desenvolvedor mantenha o foco nos requisitos e regras de negócio do seu projeto (DJANGO SOFTWARE FOUNDATION, 2017).

Meteor é um *framework* JavaScript completo, que integra as tecnologias de linguagem de programação, banco de dados e serviços Web, e tem como propósito desenvolvimento ágil. Possibilita o desenvolvimento de aplicativo para Web, Android, iOS e *desktop* (METEOR DEVELOPERS, 2017).

O presente trabalho propõe o desenvolvimento de um estudo comparativo dessas tecnologias de desenvolvimento de sistemas Web, estabelecendo as vantagens e desvantagens de cada uma das três tecnologias, bem como critérios para comparação dessas.

1.1 JUSTIFICATIVA

A tecnologia avançou com o advento da Internet e o número de tecnologias de desenvolvimento de sistemas Web também cresceu. E escolher uma tecnologia dentre tantas, não é uma tarefa tão trivial, pois essa escolha pode influenciar diretamente no sucesso do seu projeto. Essa dificuldade se acentua ainda mais

quando o desenvolvedor vai em busca de literatura que apresente comparações e informações sobre essas opções.

Este trabalho propõe um estudo comparativo das tecnologias de desenvolvimento de sistemas Web Polymer, Django e Meteor, visando indicar as vantagens e desvantagens, assim como estabelecer critérios para a escolha dessas tecnologias.

Foram escolhidas essas tecnologias, devido a questão que foi levantada no grupo de estudo de computação aplicada da UEMS (Universidade Estadual de Mato Grosso do Sul) de qual tecnologia usar em um projeto que está sendo desenvolvido pelo grupo. Então a partir de pesquisas sobre as tecnologias mais usadas atualmente, identificou-se uma carência de estudos comparativos dessas três tecnologias: Polymer, Django e Meteor. E como elas foram citadas pelos participantes do grupo, resolveu-se estuda-las.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O presente trabalho tem como objetivo geral estudar as tecnologias de desenvolvimento de sistemas Web: Polymer, Django e Meteor e seus componentes, apresentando as vantagens e desvantagens de cada uma das três tecnologias, e estabelecer critérios para a comparação dessas.

1.2.2 OBJETIVOS ESPECÍFICOS

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- Implementar um aplicativo em cada uma das três tecnologias de desenvolvimento de sistemas Web: Polymer, Django e Meteor;
- Desenvolver um roteiro de desenvolvimento de software usando essas tecnologias;
- Definir critérios e modelos de comparação;

- Comparar as três tecnologias de desenvolvimento de sistemas Web.

1.3 METODOLOGIA

Para o desenvolvimento desse projeto, foi realizada um estudo bibliográfico acerca das tecnologias de desenvolvimento Web. Foi criado um roteiro de desenvolvimento de software. Posteriormente, modelou-se um cenário que representa algumas funcionalidades de um software.

Esse cenário de estudo tem um tamanho reduzido devido ao número de implementações realizadas. Por fim, foi realizado um estudo comparativo com o objetivo de estabelecer as vantagens e desvantagens no processo de desenvolvimento em cada uma das três tecnologias.

1.4 ORGANIZAÇÃO DOS CAPÍTULOS

O Capítulo 1 contém a apresentação inicial, os objetivos, justificativa e metodologia. O Capítulo 2 apresenta definição e contextualização a respeito de sistemas de informação e algumas arquiteturas de desenvolvimento de sistemas de informação; apresenta também um levantamento teórico acerca das tecnologias abordadas e os recursos utilizados por elas. O Capítulo 3 apresenta os resultados obtidos. No Capítulo 4 encontra-se a conclusão do trabalho.

2 LEVANTAMENTO TEÓRICO

Nesse capítulo será apresentada uma análise geral dos principais pontos estudados para a realização desse trabalho.

Serão abordados os seguintes pontos: definição e contextualização a respeito de sistemas de informação e algumas arquiteturas de desenvolvimento de sistemas de informação. Serão apresentados também os recursos utilizados pelas tecnologias abordadas e a definição e características das tecnologias Polymer, Django e Meteor.

2.1 SISTEMA DE INFORMAÇÃO

De maneira crescente valoriza-se a informação como um recurso que define a competitividade de pessoas, organizações e produtos/serviços. Essa valorização, é devida ao reconhecimento de que a informação, para ser acessível, deve ser organizada e gerenciada, pois a necessidade de informação se torna cada vez mais complexa e dependente de diferentes e múltiplas fontes, cuja correta avaliação e qualidade é fator crucial para os processos de tomada de decisão (Marchiori, 2002).

Vale salientar a diferença entre dado, informação e conhecimento, sendo o contexto em que estão aplicados importante para essa diferenciação. Dados são fatos e observações brutas, entendidos como recursos de matéria-prima, não contextualizados. Informação é o resultado da análise dos dados, ou seja, é o dado processado e colocado em um contexto que lhe agregue valor para usuários finais específicos. Assim, a partir da disponibilização estruturada das informações é construído o conhecimento, um entendimento sobre situações, pessoas, objetos/produtos (Gordon; Gordon, 2006; O'Brien, 2004; Gouveia; Ranito, 2004).

O ser humano se depara com diversas situações ao longo da vida, em que precisa tomar uma decisão e para isso precisa de informação disponível de forma clara e com qualidade (Gouveia; Ranito, 2004).

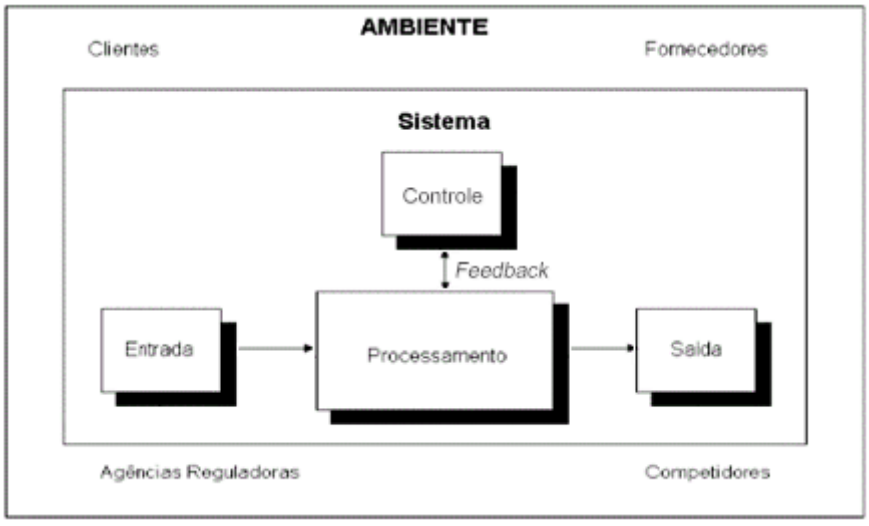
Dessa maneira, os sistemas de informação (SI) estão inseridos no contexto de encontrar a informação correta, com qualidade e de forma rápida, no entanto para entender um SI primeiramente faz-se importante entender o que é um sistema. Para Gouveia e Ranito (2004) um sistema é um conjunto de subsistemas que

interagem para alcançar um objetivo principal/geral, comum entre eles, sendo que cada subsistema tem o seu objetivo próprio que é caracterizado como um objetivo específico, requisito para o alcance do objetivo principal.

O'Brien (2004) complementa afirmando que um sistema recebe insumos e produz resultados em um processo organizado de informação, como ilustrado na Figura 1. Para isso possui as seguintes atividades básicas, a saber:

- **entrada:** consiste na captura de componentes do ambiente interno ou externo da organização, que são inseridos no sistema a fim de serem processados.
- **processamento:** utilizam processos de transformação que convertem a entrada em produto, aplicando contexto e dando significado a ela.
- **saída:** processo de transferência dos dados que foram transformados em informação para o destino final, esse destino pode ser outro sistema, um subsistema ou o usuário final.
- ***feedback:*** são dados sobre o desempenho do sistema.
- **controle:** consiste no monitoramento e avaliação do *feedback* para assim determinar se os resultados do sistema estão satisfatórios de acordo com o que foi definido como meta. E a partir dessa avaliação são feitos os ajustes necessários nas faces de entrada e processamento, para melhorar a satisfação do sistema.

Figura 1 – Atividades básicas de um Sistema



Fonte: Adaptado Laudon; Laudon (2010) e O'Brien (2004).

Diante desse conceito de sistema, podemos definir sistemas de informação como um conjunto de componentes inter-relacionados que coletam dados,

processam, armazenam esses dados e distribuem informações, com o intuito de dar suporte ao processo decisório, bem como ao gerenciamento e controle de uma determinada organização (Laudon; Laudon, 2010).

Gordon e Gordon, 2006 e Laudon e Laudon, 2010, concordam e se complementam quando descrevem os componentes de um sistema de informação. Eles destacam que um SI é mais do que computadores. Um SI é uma estrutura integrada de três dimensões, a Figura 2 apresenta as dimensões de um sistema de informação inter-relacionadas.

- **Organizações:** representa a cultura da empresa, suas regras de negócio, seus procedimentos, comportamentos e tarefas logicamente relacionados para execução do trabalho, definidos ao longo da vida da empresa.
- **Pessoas:** um sistema de informação seria inútil sem as pessoas. Como pessoas entendemos todos os atores da organização desde de o desenvolvedor do SI até utilizadores.
- **Tecnologia:** são os computadores, sistema de gestão de banco de dados, tecnologia de comunicação e redes, tecnologias específicas para atender o objetivo da organização, entre outras. O conjunto dessas tecnologias formam a estrutura de TI de uma organização, e tal estrutura deve ser bem organizada, estruturada e administrada, pois é sobre essa plataforma que o SI é montado.

Figura 2 – Dimensões de um sistema de informação.

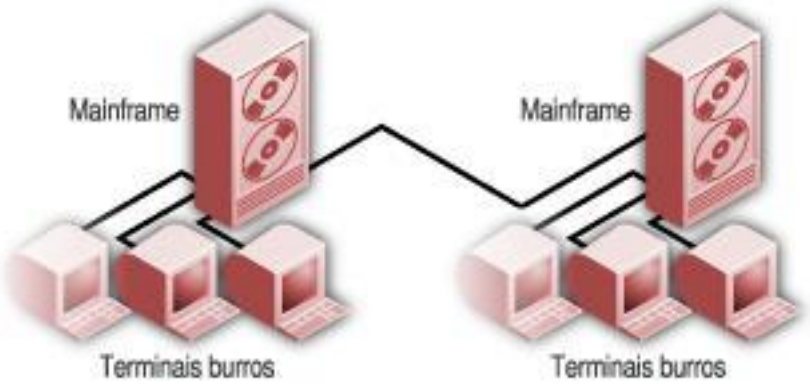
Fonte: Laudon; Laudon (2010)

Os sistemas de informação têm como objetivo nortear o processo decisório, fornecer informações através de dados brutos, cumprir os objetivos especificados e garantir a segurança, qualidade e disponibilidade dos dados e informações mediante custos aceitáveis a organização que serve (Gouveia; Ranito, 2004).

2.1.1 História dos Sistemas de Informação

Os projetos computacionais de sistemas de informação ao longo da história acompanharam a evolução dos paradigmas da informática. Na década de 1960, surgiu e se popularizou o modelo de computação centralizado, constituído por mainframes. Sistemas baseados em mainframes são compostos por “terminais burros”, ou seja, computadores que não possuem processamento (Rocha, 2002). A Figura 3 ilustra um ambiente utilizando o modelo centralizado.

Figura 3 – Paradigma centralizado.



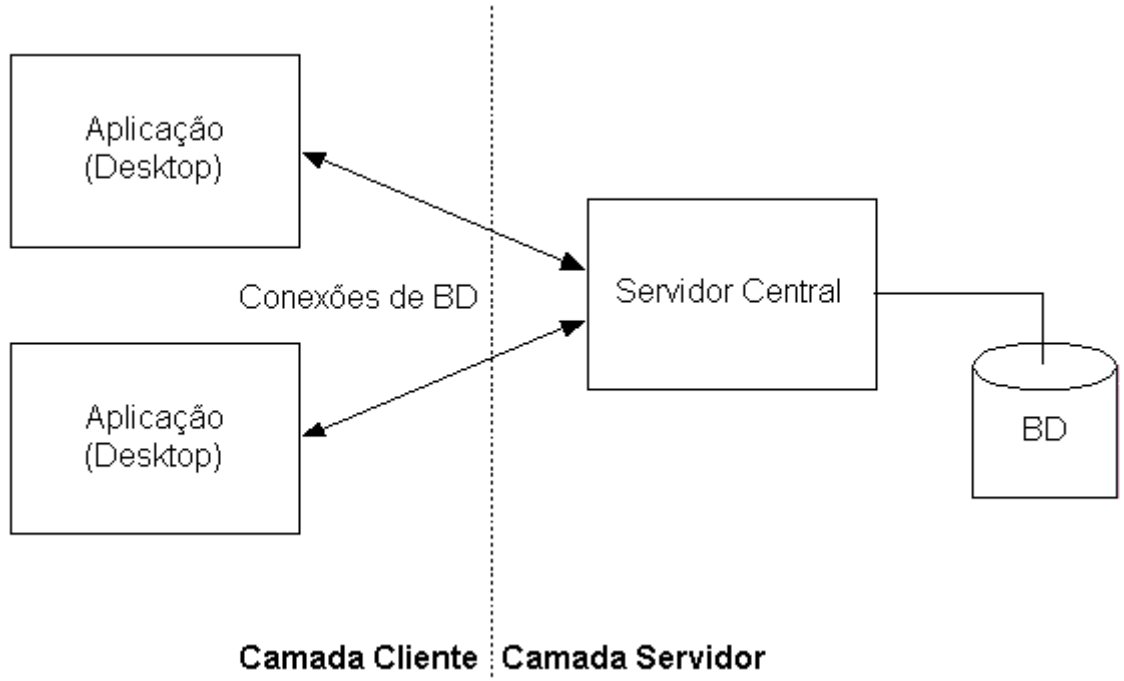
Fonte: REDES A 2001.

Seguindo na história, por volta de 1980 surgiram os computadores pessoais e juntamente com eles o paradigma cliente-servidor, que traz o conceito de dividir o processamento em duas camadas através de uma conexão, a camada cliente e a camada servidor. Cada camada geralmente encontra-se em plataformas diferentes. O cliente é composto pela interface com o usuário, navegação e apresentação do sistema de informação, e o servidor por sua vez contém o banco de dados e a lógica de manipulação dos dados (ROCHA, 2002).

Na Figura 4 podemos visualizar a modelagem de uma arquitetura cliente-servidor em 2 camadas. No entanto, nesse novo paradigma ocorreu que os clientes

ficavam muitos sobrecarregados, tornando-se necessário aumentar a capacidade dos computadores pessoais. Outra desvantagem na utilização dessa arquitetura, foi o aumento da complexidade para atualizar os clientes, pois a lógica do negócio está presente nos clientes, sendo necessário gerenciar as atualizações do software em cada cliente da rede (ROCHA, 2002).

Figura 4 – Paradigma cliente/servidor em 2 camadas

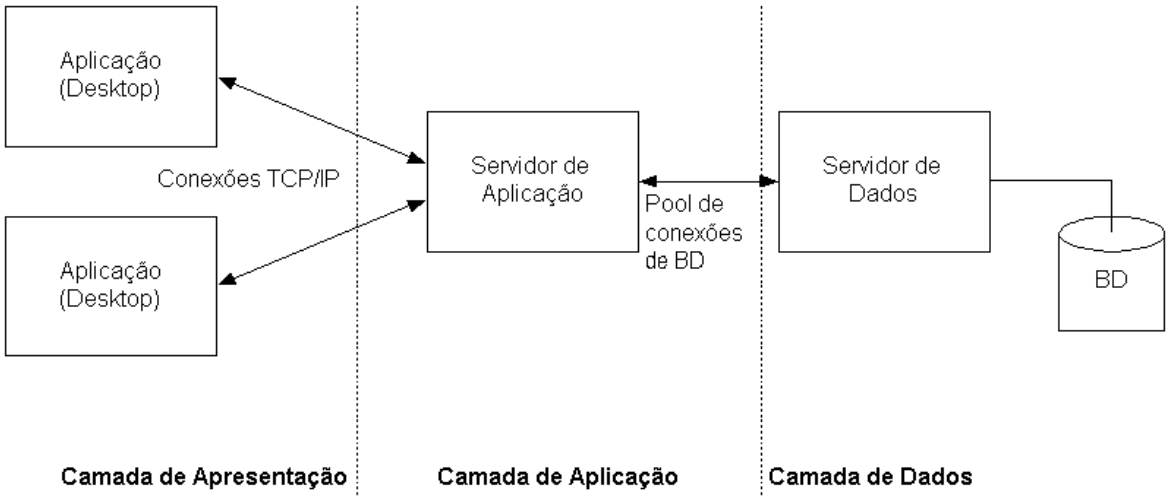


Fonte: http://www.dsc.ufcg.edu.br/~jacques/cursos/j2ee/html/intro/intro.htm.

Então surge o paradigma cliente-servidor em 3 camadas, que é uma evolução do modelo cliente-servidor em 2 camadas, onde é incluída uma nova camada entre a camada cliente e a camada servidor, esse modelo foi impulsionado pelo surgimento e crescimento da Web por volta de 1994 (ROCHA, 2002).

A camada cliente nesse paradigma é responsável pela interface gráfica com o usuário. A camada intermediaria, camada AppServer ou servidor de aplicação é responsável pelo controle da lógica de negócio e a camada servidor é constituída pelo banco de dados e pela lógica de manipulação dos dados (ROCHA, 2002). A Figura 5 apresenta a modelagem desse paradigma.

Figura 5 – Paradigma cliente/servidor em 3 camadas



Fonte: http://www.dsc.ufcg.edu.br/~jacques/cursos/j2ee/html/intro/intro.htm.

2.2 RECURSOS E CONCEITOS UTILIZADOS PELAS TECNOLOGIAS ABORDADAS

Nessa seção serão descritos os recursos e conceitos utilizados pelas tecnologias abordadas no trabalho.

2.2.1 BANCO DE DADOS

Um banco de dados é uma coleção de dados relacionados que segue algumas propriedades, a saber: os dados são derivados de alguma fonte, representam algum aspecto do mundo real e possui um grupo de usuário definido que tem interesse em seu conteúdo (ELMASRI; NAVATHE, 2005). Os dois principais modelos de banco de dados são: modelo relacional e NoSQL.

O modelo relacional representa os dados de forma estruturada, lida com os dados através de tabelas, compostas por linhas e colunas, utiliza também o conceito de restrição de integridade para garantir consistência, bem como dados altamente normalizados e utiliza o SQL como linguagem de definição, manipulação e consulta de dados (BRITO, 2010).

O termo NoSQL significa “Não apenas SQL”, é uma solução de banco de dados que aborda essencialmente os seguintes pontos: não relacional, distribuído, de código aberto, escalável horizontalmente, não possui esquema ou possui esquema flexível, acesso via API simples (NOSQL, 2017).

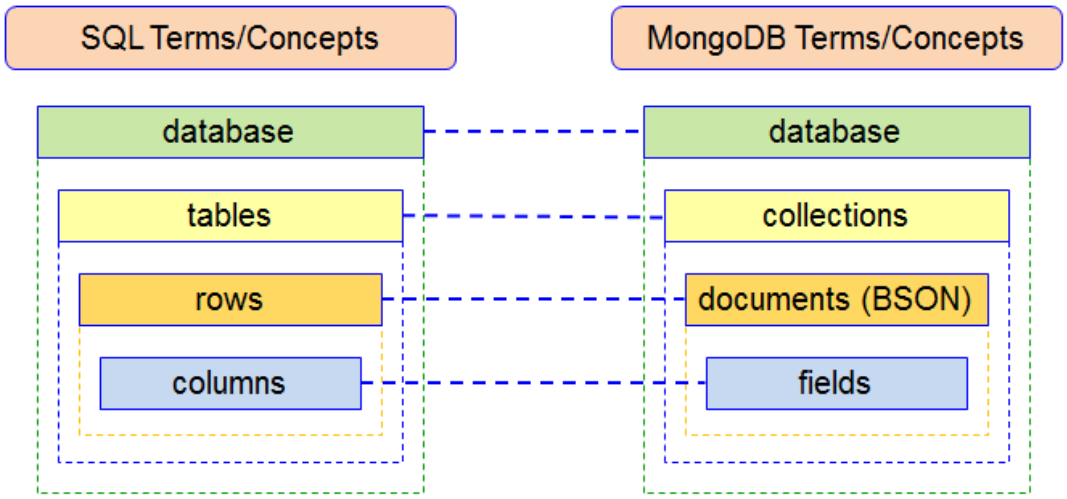
Sadalage e Fowler (2012) discordam da definição que o termo NoSQL significa “Não apenas SQL”, para eles esse significado não define a tecnologia e o importante não é saber o que significa o termo, mas sim entender o significado da tecnologia. Assim Sadalage e Fowler (2012) defendem que o termo “NoSQL” deve ser usado para se referir a um conjunto mal definido de bancos de dados principalmente de código-fonte aberto, desenvolvido em sua maioria no início do século 21, e não usando SQL principalmente.

Mesmo a definição oficial do site do NoSQL (2017) e os autores Sadalage e Fowler (2012), não concordando quanto ao significado do termo “NoSQL”, sobre as características dessa solução de banco de dados eles estão de acordo.

Bancos de dados NoSQL se destacam pela facilidade de desenvolvimento, desempenho escalável, alta disponibilidade e resiliência (AMAZON WEB SERVICES, 2017). Um banco de dados NoSQL orientado a documentos caracteriza-se pelo modo como lida com os dados. Ele trabalha com documentos autocontidos e auto descritivos e coleções de dados flexíveis, ou seja, uma coleção pode conter vários documentos com formatos diferentes, geralmente em formato XML ou JSON. (MEDEIROS, 2015; AMAZON WEB SERVICES, 2017).

A Figura 6 ilustra as principais diferenças de estrutura entre o modelo relacional e o modelo NoSQL orientado a documentos. Podemos visualizar do lado esquerdo da Figura 6 que a estrutura básica de um banco de dados relacional contém um banco de dados constituído de tabelas formadas por linhas e colunas, onde cada linha representa um registro e cada coluna representa um atributo do registro.

Figura 6 – Comparação entre o modelo relacional e modelo NoSQL orientado a documentos



Fonte: BASTOS, 2015

A direita na Figura 6 temos a estrutura básica de um banco de dados NoSQL orientado a documentos. Ela é formada por um banco de dados constituído de coleções que são conjuntos de documentos, esses documentos contêm campos definidos por uma chave e um valor, como no exemplo da Figura 7. Analogicamente, cada documento representa um registro em banco de dados relacional.

Figura 7 - Exemplo de documento NoSQL

```
1  {
2      nome: "João Silva",
3      dataNascimento: "23/11/1977",
4      sexo: "M",
5      endereco: ["Rua Tiradentes", 1891, "Bairro Central"]
6  }
```

Fonte: elaborada pela autora.

No exemplo da Figura 7, temos um documento NoSQL que estruturado através de campos divididos em chave e valor. Na linha 2 da imagem, “nome” representa a chave do campo e “João Silva” representa o valor do campo, na linha 3 “dataNascimento” representa a chave do campo e “23/11/1977” representa o valor do campo e assim sucessivamente.

MongoDB é um banco de dados NoSQL orientado a documentos, utilizado nativamente pelo *framework* Meteor. Caracteriza-se pela não definição de esquemas estáticos e a possibilidade de redundância e inconsistência, assim

como um desenvolvimento iterativo, que devido ao esquema dinâmico diminui o custo de mudanças no escopo e nos requisitos (MONGODB INC, 2017; MEDEIROS, 2015).

O MongoDB possui também modelo de dados flexível, permitindo a modificação dinâmica do seu esquema sem deixar seu banco de dados inativo. É distribuído no núcleo, o que proporciona alta disponibilidade, escala horizontal e distribuição geográfica, fáceis de construir e utilizar (MONGODB INC, 2017).

MongoDB é gratuito e de código aberto licenciado pela GNU AGPL, no entanto possui alguns serviços proprietários. Ele é escrito em C++, é multiplataforma e dá suporte a diversas linguagens de programação. O projeto MongoDB foi iniciado em 2007, tendo sua primeira versão lançada em 2009. Atualmente encontra-se na versão 4.0 (MONGODB INC, 2018; MEDEIROS, 2015).

O Meteor fornece nativamente o banco de dados MongoDB, no entanto é possível desenvolver aplicações usando outras soluções de banco de dados, nesse caso é preciso integrar ao Meteor alguns *packages* desenvolvidos pela comunidade (MAIA, 2016).

Já o Polymer possui um elemento dentro de *Data elements* chamado *polymerfire*, que implementa soluções de banco de dados baseados em Firebase (WEB COMPONENTS C, 2017).

O Django dá suporte aos bancos de dados PostgreSQL, MySQL, SQLite e Oracle (LENNON, 2009). Trabalha com ORM, Object Relational Mapping – Mapeamento objeto relacional, que é uma técnica que faz o mapeamento de um objeto para o banco de dados. O Django ORM faz o mapeamento direto, ou seja, cada linha no código é diretamente mapeada para uma linha no banco de dados e não exige prévio estabelecimento de esquema (EVERSQL TEAM, 2017).

2.2.2 HTML

HTML, *HyperText Markup Language*, em português significa, Linguagem para marcação de hipertexto. Hipertexto nesse sentido pode ser definido como conteúdo inserido em um documento Web que possibilita a interligação deste com outros documentos. HTML foi criado por Tim Berners-Lee em 1992, baseada na

especificação SGML, método internacional que especifica normas gerais para criação de linguagem de marcação (SILVA, 2011).

Com o HTML os desenvolvedores Web descrevem a estrutura do documento Web podem estruturar documentos contendo título, texto, tabelas, listas, multimídias, entre outros recursos (W3C, 2017). A Figura 8 é um exemplo de código HTML, onde podemos visualizar a estrutura básica de um documento HTML.

Figura 8 – Exemplo de código HTML

```
<!DOCTYPE html>
<html lang="pt-br">
<head> <!-- Cabeçalho -->
    <meta charset="utf-8">
    <title>Título da página</title>
</head>
<body>
    <!-- Conteúdo da Página aqui -->
    <h1>Minha primeira página HTML</h1>

    <p class="texto">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque hendrerit
    urna vitae nisi placerat, a ultricies ligula maximus. Fusce aliquet enim non
    diam sollicitudin lobortis. Maecenas eu finibus neque. Etiam luctus, arcu eget
    pharetra pulvinar, ligula velit pharetra tortor, at venenatis est sapien in leo.
    Duis a augue lobortis, pharetra.</p>
</body>
</html>
```

Fonte: elaborada pela autora.

2.2.3 CSS

CSS, *Cascading Style Sheets*, em português folhas de estilos em cascata, é uma linguagem de estilização, utilizada para descrever como as informações serão apresentadas ao usuário. Ele permite adaptar a apresentação para vários tipos de dispositivos. O CSS pode ser usado em qualquer linguagem baseada em XML (W3C, 2017). Na Figura 9 podemos visualizar um exemplo de código CSS.

Figura 9 – Exemplo de código CSS

```css
*{
    margin: 0;
    padding: 0;
    list-style: none;
}
body {
    background: #e0dedf;
    color: #333;
}
h1{
    font: 18px helvetica, sans-serif;
    text-align: center;
}
.texto{
    font: 14px arial, sans-serif;
    text-align: justify;
}
```

Fonte: elaborada pela autora.

2.2.4 JAVASCRIPT

JavaScript é uma linguagem de programação que especifica o comportamento de uma página Web. É uma linguagem de alto nível, dinâmica, interpretada e não tipada, dá suporte a estilos de programação orientados a objetos e funcionais. Foi criada na Netscape na fase inicial da Web (FLANAGAN, 2013).

Todos os navegadores modernos têm um interpretador JavaScript e desde de 2012 oferecem suporte completo ao ECMAScript 5.1 e navegadores mais antigos suportam pelo menos o ECMAScript 3 (MOZILLA FOUNDATION B, 2017).

ECMAScript foi o nome atribuído a linguagem JavaScript, quando a mesma foi submetida a ECMA – European Computer Manufacturers Association, para padronização, pois o termo “JavaScript” é uma marca registrada, licenciada pela Sun Microsystems, atualmente Oracle (FLANAGAN, 2013).

2.3 POLYMER

Polymer é uma biblioteca JavaScript desenvolvida e mantida pela Google, com o intuito de facilitar o desenvolvimento de *Web Components* (POLYMER PROJECT A, 2017). Nesse contexto, uma biblioteca é um conjunto de funções

planejadas para uma finalidade útil e geral, sendo altamente reutilizável (GAMA et al., 2000).

Web Components, por sua vez, é um conjunto de recursos para criar elementos HTML (*HyperText Markup Language* - Linguagem para marcação de hipertexto) reutilizáveis, com estilo encapsulado e comportamento personalizado, que funcionam nos navegadores modernos. Baseia-se em 4 principais especificações, a saber (WEB COMPONENTS A, 2017; WEB COMPONENTS B, 2017):

- ***Custom Elements*:** possibilita ao desenvolvedor a criação de elementos diferenciados, com estrutura, comportamento, estilo próprio e facilmente reutilizável.
- ***Shadow DOM*:** encapsula os componentes, ou seja, isola um determinado componente para que regras CSS (*Cascading Style Sheets* - Folhas de estilos em cascata) ou marcações JavaScript fora do escopo desse componente não interfiram nele.
- ***HTML Imports*:** permite encapsular e incluir documentos HTML dentro de uma página HTML.
- ***HTML Template*:** Um template contém trechos de códigos que são usados na criação de componentes dinâmicos. Ou seja, o código está presente no DOM (*Document Object Model* – Modelo de Objeto de Documento), mas só é carregado caso seja instanciado.

A combinação dessas partes possibilita a criação de componentes funcionais, portáveis, com visual e código customizados. Com *Web Components* é possível ainda integrar elementos personalizados com os elementos internos do navegador, ou então estruturar seu aplicativo de forma a criar componentes de tamanhos customizados, obtendo assim um código mais limpo e reduzindo custos de manutenabilidade (POLYMER PROJECT B, 2017).

O Polymer atualmente encontrado na versão 3.0 oferece uma coleção de elementos feitos pelos desenvolvedores da biblioteca, que podem ser usados em conjunto ou individualmente. Esses elementos encontram-se agrupados em coleções disponíveis no site da *Web Components*, a saber (WEB COMPONENTS C, 2018):

- *App elements*: elementos de aplicativo;

- *Data elements*: elementos relacionados a dados/armazenamento;
- *Gold elements*: elementos para *e-commerce*;
- *Google web componentes*: Componentes para APIs e serviços do Google;
- *Iron elements*: são os elementos centrais, o *core*;
- *Layout elements*: elementos relacionados ao *layout*;
- *Molecules*: pacotes para bibliotecas de terceiros;
- *Paper elements*: são os elementos baseados no estilo *Material Design* do Google;
- *Platinum elements*: uso *off-line*, *bluetooth*, entre outros.

Algumas das empresas que utilizam o Polymer, são: BBVA, Coca cola, Comcast, Google, ING, Net a porter, entre outros (POLYMER PROJECT B, 2017).

2.4 DJANGO

O Django é um *framework* Web de alto nível em Python, que possibilita o desenvolvimento de sites seguros, reutilizáveis, de forma rápida. Segue a política de abstrair os problemas do desenvolvimento Web, para que o desenvolvedor possa se concentrar em desenvolver seu site sem reinventar a roda (DJANGO SOFTWARE FOUNDATION, 2017).

Um *framework* é um conjunto de classes e interfaces que cooperam entre si, com o intuito de facilitar o desenvolvimento de uma categoria específica de *software* (GAMA et al., 2000; HORSTMANN, 2007). Uma característica particular dos *frameworks* é a inversão de controle, isto é, o controle do fluxo do programa encontra-se no *framework* e não no código de aplicação do desenvolvedor, ou seja, o *framework* que chama os métodos definidos pelo usuário e não o contrário. Essa característica possibilita a expansão do *framework*, pois o usuário personaliza os métodos genéricos deste (JOHNSON; FOOTE, 1988).

Django é gratuito e de código aberto. Foi inicialmente desenvolvido entre os anos de 2003 e 2005, por uma equipe de desenvolvimento responsável por um site de jornal. Essa equipe começou a identificar padrões de código e de *Design* e em julho de 2005 abriram o projeto Django. Em 2008 lançaram a versão 1.0 e atualmente encontra-se na versão 2.1.2. Possui opção de suporte gratuito e pago (MOZILLA FOUNDATION A, 2017).

Django é caracterizado por ser versátil, pois funciona com outros *frameworks* e pode fornecer conteúdo em vários formatos (HTML, XML - *eXtensible Markup Language* - Linguagem de marcação extensível, JSON - *JavaScript Object Notation* - Notação de Objetos JavaScript, entre outros). Oferece autenticação de usuário, proteção contra injeção de SQL, falsificação de solicitação entre sites, entre outras opções de segurança. Possui uma arquitetura baseada em componentes, o que facilita a manutenção e compreensão do código (DJANGO SOFTWARE FOUNDATION, 2017; MOZILLA FOUNDATION A, 2017).

O *framework* utiliza o padrão MTV (*Model-Template-View*), *Model* é a camada de acesso aos dados, *Template* define como os dados devem ser apresentados ao usuário e *View* é a camada que descreve quais dados devem ser apresentados ao usuário. É portável, ou seja, é executado em várias plataformas como: Linux, Windows e Mac OS X (MOZILLA FOUNDATION A, 2017; LENNON, 2009).

Algumas das empresas que utilizam o Django, são: Disqus, Fundação MacArthur, Open Knowledge Foundation, Instagram, Mozilla, National Geographic, Pinterest e Open Stack (DJANGO SOFTWARE FOUNDATION, 2017).

2.5 METEOR

Meteor é um *full-stack* JavaScript *platform*, em tradução livre Plataforma de JavaScript de pilha completa, ou seja, uma plataforma completa que integra linguagem de programação, banco de dados e serviços Web. Tem o propósito de facilitar o desenvolvimento de aplicações Web e móveis de forma ágil (METEOR DEVELOPERS, 2017).

Meteor foi desenvolvido baseado em Node.js que consiste em uma plataforma projetada para desenvolver aplicações de rede rápidas e escaláveis, funciona sobre V8 JavaScript do Chrome, usa *I/O* (Input/Output – entrada/saída) não bloqueante, o que o torna leve e eficiente, e é orientado a eventos (ABERNETHY, 2011; NODE.JS, 2017).

Meteor utiliza a linguagem de programação Javascript como padrão tanto no cliente como no servidor e HTML e CSS para criar interface de usuário. Possibilita o compartilhamento de código entre o cliente e o servidor, e o acesso ao banco de

dados pode ser realizado pelos dois ambientes (COLEMAN, GREIF, 2013; WODEHOUSE, 2016).

O Meteor envia dados pela rede ao invés de código HTML, e esses dados são processados no cliente, possibilita também a criação de aplicações iOS, Android ou Web com o mesmo código através da integração com o Apache Cordova. Meteor é reativo, pois realiza atualização de dados automaticamente, ou seja, caso ocorra uma atualização do banco de dados, essa atualização é refletida na interface do usuário automaticamente, sem precisar da interferência do usuário (METEOR DEVELOPERS, 2017; WODEHOUSE, 2016).

O Meteor está disponível sobre a licença MIT. O projeto Meteor foi fundado em 2011 e encontra-se atualmente na versão 1.7.0.3. Algumas das empresas que utilizam o Meteor, são: Mazda, Ikea, Honeywell, Qualcomm, PGA Tour (METEOR DEVELOPERS, 2017; EMPSON, 2012).

3 RESULTADOS

Nessa seção serão descritos os pontos observados de cada tecnologia de desenvolvimento de sistemas Web estuda. Os aplicativos desenvolvidos implementaram as funcionalidades descritas no apêndice A.

3.1 POLYMER

Polymer é uma biblioteca JavaScript de código-fonte aberto desenvolvida com o intuito de facilitar a criação de *Web Components.* No entanto, existem alguns aspectos negativos apresentados pela biblioteca que impactam principalmente sobre os iniciantes. A falta de documentação é um desses aspectos negativos, nem todos os elementos do Polymer são documentados e às vezes a única orientação sobre como usá-los é o código de demonstração do elemento.

A forma como deve-se organizar os aplicativos também não é muito clara. A comunidade não é tão ativa se comparada com comunidades de outras bibliotecas e *frameworks* Javascript. Para iniciantes em *Web Components*, existe uma curva de aprendizado considerável, o torna o processo de desenvolvimento mais lento.

Para persistência de dados o Polymer possui dentro de Data *elements* o chamado *polymerfire*, que implementa soluções de bancos de dados baseados em Firebase. Possui muitos componentes, disponibilizados pela comunidade e pela equipe Polymer, mas apresenta problema de dependência na utilização desses componentes. A atualização de conteúdo depende da ação do usuário.

Gray Norton (2018), o líder tecnológico do projeto Polymer, em uma discussão no GitHub do Polymer, anunciou que a biblioteca Polymer, bem como suas ferramentas, tais como Polymer *Elements*, Polymer *App Toolbox* e o Polymer CLI estão entrando em modo de manutenção, isso significa que a equipe Polymer continuará a dar suporte a correções de bugs críticos e versões de manutenção, no entanto não serão desenvolvidos novos recursos ativos.

A equipe do Polymer passará a construir uma coleção de ferramentas utilizáveis separadamente. A recomendação para novos projetos é construir sobre o *LitElement*, que é “uma classe base simples para criar elementos customizados

renderizados com o *lit-html*" (WEB COMPONENTS D, 2018). *Lit-html* por sua vez "é uma biblioteca de *templates* pequena e leve. O *Lit-html* usa *strings* de modelo JavaScript para criar modelos expressivos e dinâmicos" (RYLAN, 2018).

3.1.1 Roteiro de desenvolvimento Polymer

O roteiro de desenvolvimento do Polymer não foi desenvolvido, devido à dificuldade de encontrar documentação sobre os elementos do Polymer, bem como a dificuldade de organização do projeto que não é muito clara, ficando difícil para iniciantes saber onde cada elemento deve ser desenvolvido e como deve ser usado.

3.2 DJANGO

O Django é um *framework* Web de alto nível em Python, que possibilita o desenvolvimento Web rápido e reutilizável. Sua rapidez de desenvolvimento se deve aos vários recursos que traz consigo.

Tem uma comunidade muito grande e ativa. Apresenta uma documentação atualizada e clara, com definições e exemplos. E apresenta uma boa estrutura de organização base dos projetos, dividindo o projeto em aplicações tornando-o fácil de compreender e integrar com projetos de terceiros e de suas aplicações em outros projetos.

A gestão e persistência de dados no Django é feita de forma simples através do ORM que é uma técnica de mapeamento objeto relacional que permite relacionar os objetos com os dados que os mesmos representam. Bem como, pelo painel administrativo, com o qual a criação, edição e atualização dos modelos de dados torna-se mais fácil. A atualização de conteúdo depende da ação do usuário e apresenta uma alta reusabilidade, devido a estruturação em aplicações.

Sua arquitetura MTV (*Model-Template-View*), que divide a lógica e a visualização do sistema possibilita um desenvolvimento mais simples, independente da complexidade da aplicação.

O Django disponibiliza um sistema de *templates* que possibilita a criação de um *template* base com uma estrutura básica do seu sistema e vários *templates*

específicos que herdam as configurações do *template* base e atribuem suas configurações especificas.

Seguindo a filosofia "*Don't Repeat Yourself*" em tradução literal "Não se repita", o Django traz vários recursos inclusos por padrão, alguns exemplos são: o painel administrativo, com o qual a criação, edição e atualização dos modelos de dados torna-se mais fácil. O módulo de autenticação e permissões do usuário que realiza toda a autenticação e permissão de usuários com segurança. O framework inclui também formulários e roteamentos.

Atualmente, o Django encontra-se em uso com atualizações de correção e aprimoramento.

3.2.1 Roteiro de desenvolvimento Django

Nessa seção será apresentado um roteiro de desenvolvimento de sistemas Web utilizando o *framework* Django.

Instalação

Para utilizar o Django é necessário a instalação dos seguintes requisitos:

- Python 3 ou superior.
- PIP, gerenciador de pacotes do Python.

Depois da instalação desses requisitos, o Django é instalado através do PIP:

pip install django

Desenvolvimento

Passo 1: Criação do projeto Django.

O comando para criação do projeto Django é o seguinte:

django-admin startproject <nome-projeto>

Após criar o projeto, abra o projeto em um editor de código de sua preferência. Nesse momento configuramos o banco de dados desejado, o Django dá suporte PostgreSQL, MySQL, SQLite e Oracle.

Passo 2: Criação da aplicação.

O Django irá criar uma pasta com o nome da aplicação e alguns arquivos *default*, como models.py, view.py, admin.py, entre outros.

python manage.py startapp <nome-app>

Passo 3: Registrar aplicação em “INSTALLED_APPS” no arquivo “settings.py”.

Para que o Django reconheça sua aplicação é preciso registrar sua aplicação no arquivo de configuração “settings.py”.

Passo 4: Criação dos *Models*.

Na pasta da aplicação, no arquivo “models.py” devem ser criados os modelos de dados, que através do ORM serão mapeados para o banco de dados escolhido.

Depois que terminar as modificações no “models.py”, deve ser realizada a migração dos dados para o banco de dados.

(1) python manage.py makemigrations <nome-app>

(2) python manage.py migrate

O comando (1) é responsável por criar novas migrações com base nas alterações feitas nos *models*. O comando (2) é responsável por aplicar as migrações criadas pelo comando (1).

Passo 5: Registrar *models* no arquivo “admin.py”.

Para que os *models* possam ser gerenciados pelo painel administrativo do Django, no arquivo “admin.py” os *models* devem ser adicionados, da seguinte forma:

admin.site.register(<nome_model>)

Para esse comando funcionar corretamente, os *models* a serem registrados devem ser importados no topo da página. Agora antes de acessar o painel administrativo do Django pela primeira vez, é preciso criar um usuário.

python manage.py createsuperuser

Passo 6: Criação das funções que definem as regras de apresentação.

No arquivo “views.py” são definidas funções que recebem uma requisição (request), processam essa requisição de acordo com as regras de negócio e retornam um HTTPResponse.

Passo 7: Criação de rotas.

No arquivo “urls.py”, é definido uma ou mais rotas para cada função. Para isso importamos o arquivo “views.py” e definimos uma rota da seguinte forma.

path(‘posts/’, views.post_list)

No exemplo, está sendo definido que quando o usuário tentar acessar a rota ‘posts/’, o Django chama a função corresponde, no caso ‘views.post_list’ e retorna a resposta HTML definida na função.

Passo 8: Criação dos templates.

Para usar o sistema de *templates* do Django, cria-se uma pasta com o nome da preferência do desenvolvedor, para esse exemplo irei nomear a pasta de “templates”. Depois no arquivo “settings.py” em “TEMPLATES” definimos o caminho da pasta criada.

Passo 9: Criação do arquivo base do *template*.

Na pasta “templates” criada anteriormente, criamos um arquivo .html com o nome da preferência do desenvolvedor. Nesse arquivo, serão especificadas todas as configurações gerais do *layout* do sistema, configurações essas que serão comuns a maioria das páginas.

Passo 10: Criação dos arquivos específicos do *template*.

Para criar as demais páginas é necessário apenas estender o arquivo base do *template* e modificar somente as partes específicas. Assim caso haja necessidade de modificar o layout do sistema a modificação ocorrerá em apenas um lugar.

Passo 11: Formulários.

Para utilizar os formulários do Django, no arquivo “forms.py” é preciso criar uma classe que define o *model* a que o formulário irá se referir e os campos do *model* referido que serão utilizados no formulário.

O Django oferece um servidor web de desenvolvimento e para utilizá-lo no diretório do projeto execute:

python manage.py runserver

3.3 METEOR

O Meteor é um *framework* completo que usa Javascript como linguagem universal, ou seja, tanto no cliente, como no servidor. Essa linguagem única em ambos os lados, possibilita acesso direto ao banco de dados. Oferece uma documentação clara e convenções de programação bem estabelecidas, o que facilita o aprendizado para iniciantes. Tem uma comunidade ativa.

O desenvolvedor tem a liberdade de dividir o código da forma que melhor lhe convier, podendo decidir quais funções devem ser executadas no lado do cliente e quais no servidor. A grande quantidade de pacotes e a característica reativa, agiliza o desenvolvimento, mas ainda é preciso dedicar um tempo para persistência de dados, tempo esse maior que no Django.

O Meteor conta com uma grande quantidade e variedade de pacotes que podem ser instalados e usados em seus projetos, tanto disponibilizados pela comunidade como pela própria equipe do Meteor, esse aspecto facilita muito o desenvolvimento.

Meteor foi projetado com o intuito de facilitar o desenvolvimento de aplicações com atualização de conteúdo automática para Web e *mobile*. Possibilita aplicações Web em tempo real automaticamente, não sendo necessário fazer configurações adicionais para ativar essa funcionalidade. Funciona baseado no NodeJS.

Sobre a reusabilidade no Meteor ela depende do desenvolver. Atualmente, o Django encontra-se em uso com atualizações de correção e aprimoramento, em crescimento.

3.3.1 Roteiro de desenvolvimento Meteor

Nessa seção será apresentado um roteiro de desenvolvimento de sistemas Web utilizando o *framework* Meteor.

Instalação

Para utilizar o Meteor é necessário a instalação dos seguintes requisitos:

- NodeJS.
- Chocolaty.
- MongoDB.

Depois da instalação desses requisitos, no Windows o Meteor é instalado através do chocolaty:

choco install meteor

Desenvolvimento

Passo 1: Criação do projeto Meteor.

O comando para criação do projeto Django é o seguinte:

meteor create <nome_projeto>

Após criar o projeto, abra-o em um editor de código de sua preferência. Na estrutura de pastas do projeto, terá uma pasta nomeada “client” e uma nomeada “server”. Dentro da pasta “client” devem ser colocados todos os arquivos referentes ao lado do cliente. Dentro da pasta “server” devem ser colocados todos os arquivos referentes ao lado do servidor. Esses arquivos podem ser .js, .html e .css.

Passo 2: Criação do *template* dinâmico.

No lado do cliente, em um arquivo .html, deve-se criar um *template* dinâmico, da seguinte forma:

{{>Template.dynamic template=content}}

Passo 3: Instalar pacotes.

Para acessarmos os *templates*, precisamos denominar rotas e para isso precisamos instalar o *Flow Router* que é um pacote de roteamento da comunidade para o Meteor. Para utilizar o *Flow Router* precisamos instalar os seguites pacotes:

(1) meteor add kadira:flow-router

(2) meteor add kadira:blaze-layout

O comando (1) instala o pacote que é responsável por fazer o roteamento do lado do cliente. E o comando (2) instala um pacote que faz o gerenciamento de layout projetado para o *Blaze* e integrar o *Blaze* e o *Flow Router*.

Passo 4: Criação do *template* específicos.

No arquivo .html, do lado do cliente, é criado um *template* para cada funcionalidade.

Passo 5: Criação de rotas.

Do lado do cliente, em um arquivo .JS, é criada uma rota com o *Flow Route*, da seguinte forma.

```
FlowRouter.route('/post', {
    action: () => {
        BlazeLayout.render('main', {
            content: 'listPost'
        });
    }
});
```

Passo 6: Criação coleções.

O Meteor por padrão utiliza o banco de dados MongoDB. Então para persistir os dados, é preciso criar coleções no mongoDB. Para isso é necessário importar, o mongo do Meteor no topo da página. Podem ser criadas quantas coleções forem necessárias.

```
(1) import { Mongo } from 'meteor/mongo';
(2) const Categoria = new Mongo.Collection('categorias');
```

O comando (1) é a importação do mongo. E o comando (2) é a criação da coleção.

Passo 7: Criação de *helpers*.

Ainda no arquivo .JS, devemos criar *helpers* para os *templates*, da seguinte forma:

```
Template.listPost.helpers({
    listPost: () => {
       return Post.find();
    }
});
```

No exemplo, quando o usuário acessar a rota ‘/post’, o *Flow Router* irá renderizar o *template* ‘listPost’, e o *helper* do exemplo acima será disparado fazendo uma pesquisa no banco de dados, buscando por todos os posts cadastrados.

Passo 8: Criação de eventos.

Ainda no arquivo .JS, devemos criar eventos para os *templates*, da seguinte forma:

```
Template.addCategoria.events({
    'click #saveCategoria': (event, template) => {
        event.preventDefault();
        let categoria = {
            nome: template.find('input[name="nome"]').value
        };
        Categoria.insert(categoria);
        FlowRouter.go('/categoria');
    }
});
```

No exemplo, quando o usuário clicar no elemento que tem o id ‘saveCategoria’, no template ‘addCategoria’, o evento do exemplo será disparado e a categoria será salva no banco de dados.

O Meteor oferece um servidor web de desenvolvimento e para utilizá-lo no diretório do projeto execute:

```
meteor
```

O Quadro 1, apresenta uma comparação resumida das três tecnologias estudadas.

Quadro 1 – Comparação das três tecnologias estudadas: Polymer, Django e Meteor.

Critérios	Polymer	Django	Meteor
Documentação	Pouca	Muita	Suficiente
Comunidade	Pouco ativa	Ativa	Ativa
Organização dos projetos	Não é muito clara	Boa, divide os projetos em aplicações	Boa, o desenvolver tem liberdade de escolha
Agilidade de desenvolvimento	Mediana	Muita	Mediana
Gestão e persistência de dados	Elemento *polymerfire,* baseado em Firebase	ORM e painel administrativo	NoSQL, mongoDB por padrão
Pacotes e Componentes	Muitos	Suficiente	Muitos
Atualização de dados	Depende da ação do usuário	Depende da ação do usuário	Automática
Reusabilidade	Alta, conceito de *Web Components*	Alta, divide o projeto em aplicações	Pouca, depende do desenvolvedor
Plataforma	Web	Multiplataforma	Multiplataforma
No mercado	Em desuso, em fase manutenção	Em uso, com atualizações	Em uso, com atualizações

Fonte: elaborada pela autora.

4 CONCLUSÃO

A Web está em constante evolução e essa evolução impulsiona o aprimoramento e criação de novas tecnologias de desenvolvimento Web que sejam fáceis e rápidas de usar, e que disponibilize o maior número de recursos da melhor forma possível aos desenvolvedores. Neste trabalho apresentamos um estudo de três tecnologias de desenvolvimento de sistemas Web, são elas: Polymer, Django e Meteor.

Para realizarmos esse estudo, realizamos um estudo bibliográfico acerca das tecnologias de desenvolvimento Web, a fim de adquirimos conhecimentos para implementar um aplicativo em cada uma dessas tecnologias. E durante esse processo de estudo e implementação foi possível analisar as tecnologias e suas características, vantagens e desvantagens.

Percebeu-se que no desenvolvimento com dependência um aspecto que se deve ficar atento é com os módulos adicionais que são necessários durante o desenvolvimento, pois atualizações e compatibilidade são pontos sensíveis.

Observou-se que a tecnologia Polymer está entrando em modo de manutenção, o que significa que não é recomendável que se inicie um projeto com essa tecnologia a partir de agora, pois a equipe do Polymer só irá corrigir erros críticos para que os sistemas que já foram implementados possam continuar funcionando.

A tecnologia Django é um *framework* que está há bastante tempo no mercado, então já está bem consolidado, apresenta uma boa documentação e comunidade ativa. Tem uma estrutura bem organizada que possibilita fácil compreensão de projeto, bem como integração com pacotes de terceiro e fácil reuso. Traz também vários recursos embutidos o que torna o desenvolvimento ágil.

O Meteor é uma tecnologia relativamente nova lançada em 2011, mas que traz consigo vários recursos muito valiosos e úteis, tais como atualização de conteúdo em tempo real por padrão, sem o desenvolvedor precisar sincronizar o código, linguagem de programação única no lado do cliente e no servidor, vários pacotes com funcionalidades úteis, facilmente instalados no projeto. Possui uma documentação clara e uma comunidade ativa.

REFERÊNCIAS BIBLIOGRÁFICAS

ABERNETHY, Michael. Just what is Node.js. **IBM Developer Works**. 2011. Disponível em: <https://kyulingcompany.files.wordpress.com/2012/06/os-nodejs-pdf.pdf>. Acesso em: 05 set. 2017.

AMAZON Web Services. **O que é NoSQL?** Bancos de dados não relacionais de alto desempenho. Disponível em: <https://aws.amazon.com/pt/nosql/>. Acesso em: 20 set. 2017.

BASTOS, Thalles. **Iniciando com o MongoDB** - Diferenças entre SQL e NoSQL. 2015. Disponível em: <http://thbastos.com/blog/iniciando-com-o-mongodb-diferencas-entre-sql-e-nosql>. Acesso em: 10 set. 2017.

BRITO, Ricardo W. **Banco de dados NoSQL x SGBDs relacionais**: análise comparativa. 2010. Disponível em: <http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf>. Acesso em: 10 set. 2017.

COLEMAN, Tom; GREIF, Sacha. **Descubra Meteor**. 2014. Disponível em: <http://pt.discovermeteor.com/chapters/introduction/>. Acesso em: 05 set. 2017.

DANAILOV, Dimitar. **Polymer vs Angular vs React:** Using Web Components to Evolve the Experience. 2016. Disponível em: <https://mentormate.com/blog/polymer-vs-angular-future-web-apps/?utm_source=Medium&utm_campaign=10-14-16 Polymer Lang Translation Inquiry&utm_medium=Social>. Acesso em: 30 out. 2017.

DJANGO Software Foundation. **Django documentation**. Disponível em: <https://www.djangoproject.com/start/overview/>. Acesso em: 04 set. 2017.

ELMASRI, Ramez; NAVATHE, Shamkant B. **SISTEMAS DE BANCO DE DADOS**. São Paulo: Pearson Addison Wesley, 2005.

EMPSON, Rip. **Meteor:** Etherpad Founder & Other Rockstars Team Up To Make Web App Development A Breeze. 2012. Disponível em: <https://techcrunch.com/2012/04/11/meteor-launch/>. Acesso em: 09 nov. 2017.

EVERSQL TEAM. **Django vs SQLAlchemy** – Which Python ORM is better? 2017. Disponível em: <https://www.eversql.com/django-vs-sqlalchemy-which-python-orm-is-better/>. Acesso em: 11 nov. 2017.

FLANAGAN, David. **JavaScript**: o guia definitivo. 6. ed. Porto Alegre: Bookman, 2013.

GAMA, Erich et al. **Padrões de projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

GORDON, Steven R.; GORDON, Judith R. **Sistemas de informação**: uma abordagem gerencial. 3. ed. LTC, 2006.

GOUVEIA, Luís B.; RANITO, João. **Sistemas de informação de apoio à gestão**. Porto, 2004.

HORSTMANN, Cay. **Padrões e projeto orientado a objetos.** Porto Alegre: Bookman, 2007.

JOHNSON, Ralph E.; FOOTE, Brian. Designing Reusable Classes. **Journal Of Object-oriented Programming.** Illinois, p. 22-35. jul. 1988. Disponível em: <http://www.laputan.org/drc/drc.html>. Acesso em: 04 set. 2017.

LAUDON, Kenneth.; LAUDON, Jane. **Sistema de informação gerenciais**. 9. ed. Pearson Prentice Hall, 2010.

LENNON, Joe. **Implementando Aplicativos Django em um Servidor de Produção.** 2009. Disponível em: <https://www.ibm.com/developerworks/br/library/os-django/>. Acesso em: 21 set. 2017.

MAIA, Frederico. **Afinal, o que é Meteor?.** 2016. Disponível em: <http://f7labs.com.br/afinal-o-que-e-meteor/>. Acesso em: 06 set. 2017.

MARCHIORI, Patrícia Z. **A ciência e a gestão da informação:** compatibilidades no espaço profissional. Ciência da informação, v. 31, n. 2, 2002.

MEDEIROS, Higor. **Introdução ao MongoDB.** 2015. Disponível em: <http://www.devmedia.com.br/introducao-ao-mongodb/30792>. Acesso em: 10 set. 2017.

METEOR developers. **Introducing Meteor API Docs**. Disponível em: <http://docs.meteor.com/#/full/>. Acesso em: 04 set. 2017.

MINGOIA, Alex. **Here's the difference between Polymer and Angular**. 2014. Disponível em: <http://www.binpress.com/blog/2014/06/26/polymer-vs-angular/>. Acesso em: 30 out. 2017.

MONGODB INC. **MongoDB Architecture**. Disponível em: <https://www.mongodb.com/mongodb-architecture>. Acesso em: 11 out. 2018.

MOZILLA Foundation A. **Django introduction**. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction>. Acesso em: 04 set. 2017.

MOZILLA Foundation B. **JavaScript**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 20 set. 2017.

NODE.JS. **About Node.js**. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 05 set. 2017.

NORTON, Gray. **Polymer 4 and beyond:** what will be deprecated, and what will stay?. Disponível em: <https://github.com/Polymer/polymer/issues/5240#issuecomment-395922668>. Acesso em: 13 out. 2018.

NOSQL. Disponível em: <http://nosql-database.org/>. Acesso em: 2 0 set. 2017.

O'BRIEN, James A. **Sistemas de informação e as decisões gerenciais na era da Internet**. 2. ed. São Paulo: Editora Saraiva, 2004.

POLYMER project A: about the Polymer project. Disponível em: <https://www.polymer-project.org/about>. Acesso em: 25 ago. 2017.

POLYMER project B. Disponível em: <https://www.polymer-project.org/>. Acesso em: 25 ago. 2017.

REDES A 2001. **Redes de Computadores.** Disponível em: <http://redesa2001rj.tripod.com/dicas/redes_de_computadores.htm>. Acesso em: 09 nov. 2017.

RYLAN, Cory. Building **Web Components with lit-html**. 2018. Disponível em: <https://coryrylan.com/blog/building-web-components-with-lit-html>. Acesso em: 17 out. 2018.

ROCHA, Carlos André de Sousa. **Análise de Desempenho Em Ambientes Cliente/Servidor 2-Camadas E 3-Camadas.** 2002. 149 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal de Santa Catarina Programa de Pós-graduação em Ciência da Computação, Florianópolis, 2002. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/84180/190171.pdf?sequence=1>. Acesso em: 06 nov. 2017.

SADALAGE, Pramod J.; FOWLER, Martin. **NoSQL Distilled:** A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, 2013.

SIDORENKO, Vladimir. **7 Best Frameworks For Web Development in 2017**. 2017. Disponível em: <https://gearheart.io/blog/7-best-frameworks-for-web-development-in-2017/>. Acesso em: 30 out. 2017.

SILVA, Maurício Samy. **HTML 5:** A linguagem de marcação que revolucionou a Web. São Paulo: Novatec, 2011. 320 p.

STRAHAN, Ben**. MeteorJS vs AngularJS ain't a thing**. 2015. Disponível em: <https://medium.com/@benstr/meteorjs-vs-angularjs-aint-a-thing-3559b74d52cc>. Acesso em: 30 out. 2017.

URDHWARESHE, Rucha. **Polymer vs. React**: Comparison between Two Front End Javascript Libraries. 2016. Disponível em: <http://www.cuelogic.com/blog/polymer-vs-react-comparison-between-two-front-end-javascript-libraries/>. Acesso em: 30 out. 2017.

W3C. **HTML e CSS**. Disponível em: <https://www.w3.org/standards/webdesign/htmlcss>. Acesso em: 14 set. 2017.

WEB Components A. **Introduction**. Disponível em: <https://www.webcomponents.org/introduction>. Acesso em: 26 ago. 2017.

WEB Components B. **Specifications**. Disponível em: <https://www.webcomponents.org/specs>. Acesso em: 26 ago. 2017.

WEB Components C. **Polymer Catalog**. Disponível em: <https://www.webcomponents.org/collection/Polymer/elements>. Acesso em: 26 set. 2018.

WEB Components D. **LitElement**. Disponível em: <https://www.webcomponents.org/element/@polymer/lit-element>. Acesso em: 17 out. 2018.

WODEHOUSE, Carey. **Is MeteorJS the Right Node.js Framework for Your App?.** 2016. Disponível em: <https://www.upwork.com/hiring/development/meteorjs-node-js-framework/>. Acesso em: 06 set. 2017.

APÊNDICE A – Cenário de estudo.

O cenário utilizado para fazer a comparação das tecnologias de desenvolvimento de sistemas Web foi um blog, com as seguintes funcionalidades: postagem de artigos, possibilidade de comentar os artigos e divisão dos artigos por categoria.

No blog, usuário administrador, pode postar artigos e atribuir categorias para as postagens e usuários comuns podem ler as postagens e comenta-las, caso queiram. O blog tem os seguintes requisitos:

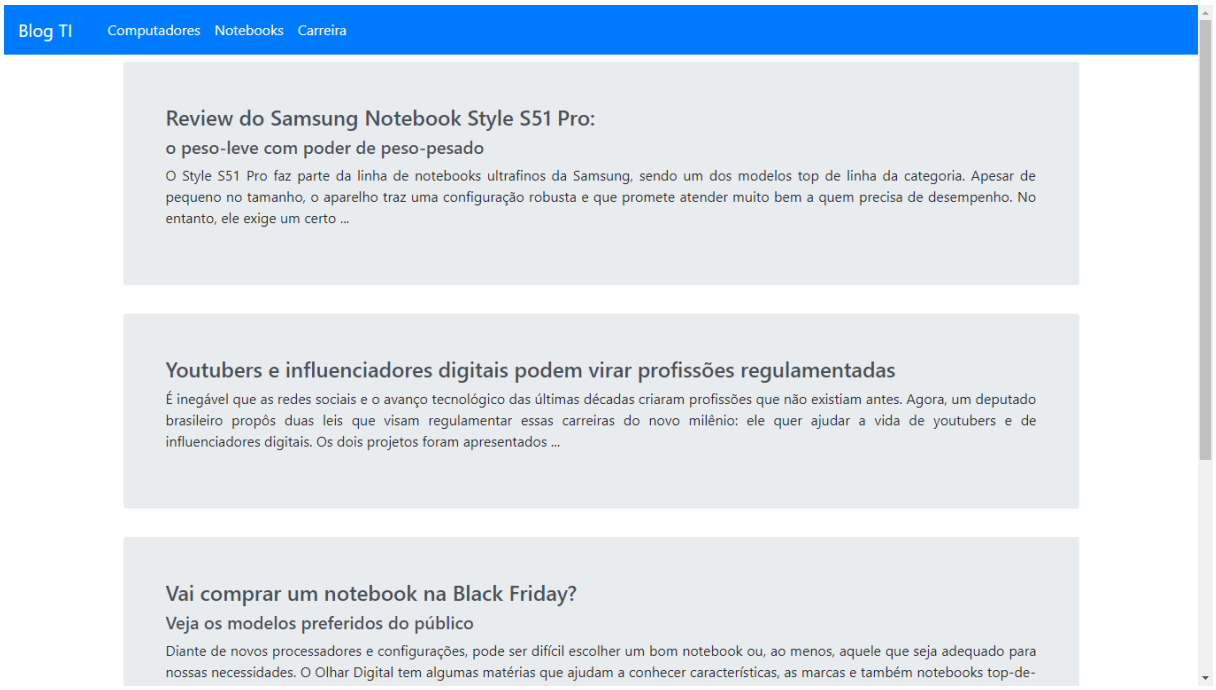
1. O sistema permite o gerenciamento das postagens de artigos.
2. O sistema permite o gerenciamento das categorias.
3. O sistema permite visualizar as postagens.
4. O sistema permite o realizar comentários nas postagens.
5. O sistema permite pesquisar as postagens por categorias.
6. O sistema tem dois tipos de usuários com níveis de acesso diferentes, são eles: o usuário administrador que tem acesso total ao sistema e o usuário leitor que pode ler e comentar as postagens.

No cenário implementado, uma postagem pode ter várias categorias e uma categoria pode pertencer à várias postagens. Assim como uma postagem deve ter autor e um autor pode fazer várias postagens.

Um comentário é realizado para apenas uma postagem, no entanto uma mesma postagem pode ter vários comentários. Para fazer a leitura e comentar as postagens o usuário não precisa necessariamente se identificar.

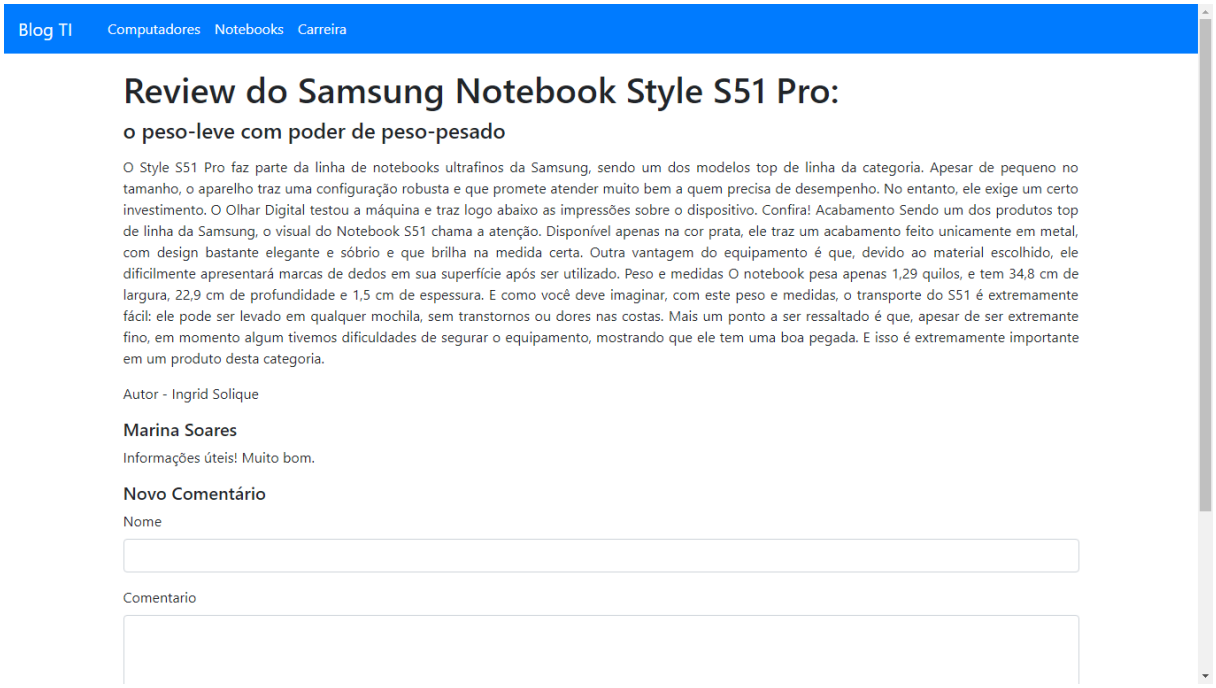
O cenário descrito acima desenvolvido na tecnologia Django, gerou o sistema ilustrado pelas imagens a seguir.

Figura 10 – Página inicial do Blog com Django, apresenta a listagem dos artigos



Fonte: elaborada pela autora.

Figura 11 – Página de visualização dos artigos com Django, apresenta todo o conteúdo do artigo e os comentários – Parte 1

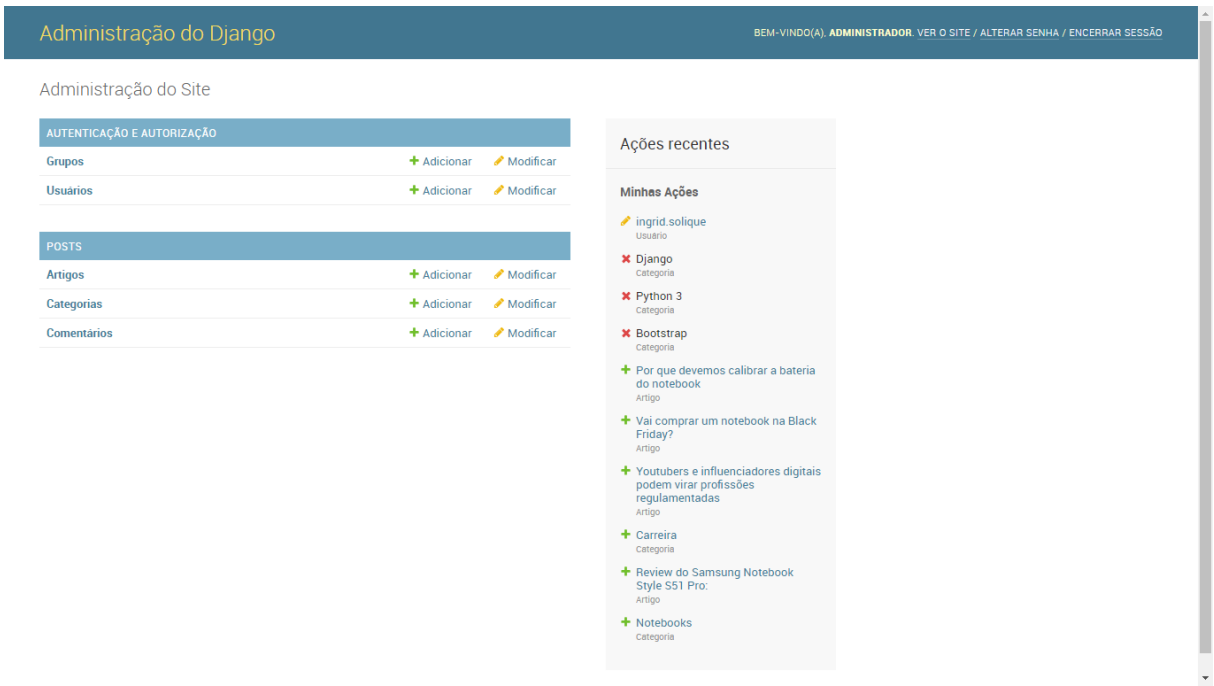


Fonte: elaborada pela autora.

Figura 12 – Página de visualização dos artigos com Django, apresenta todo o conteúdo do artigo e os comentários – Parte 2

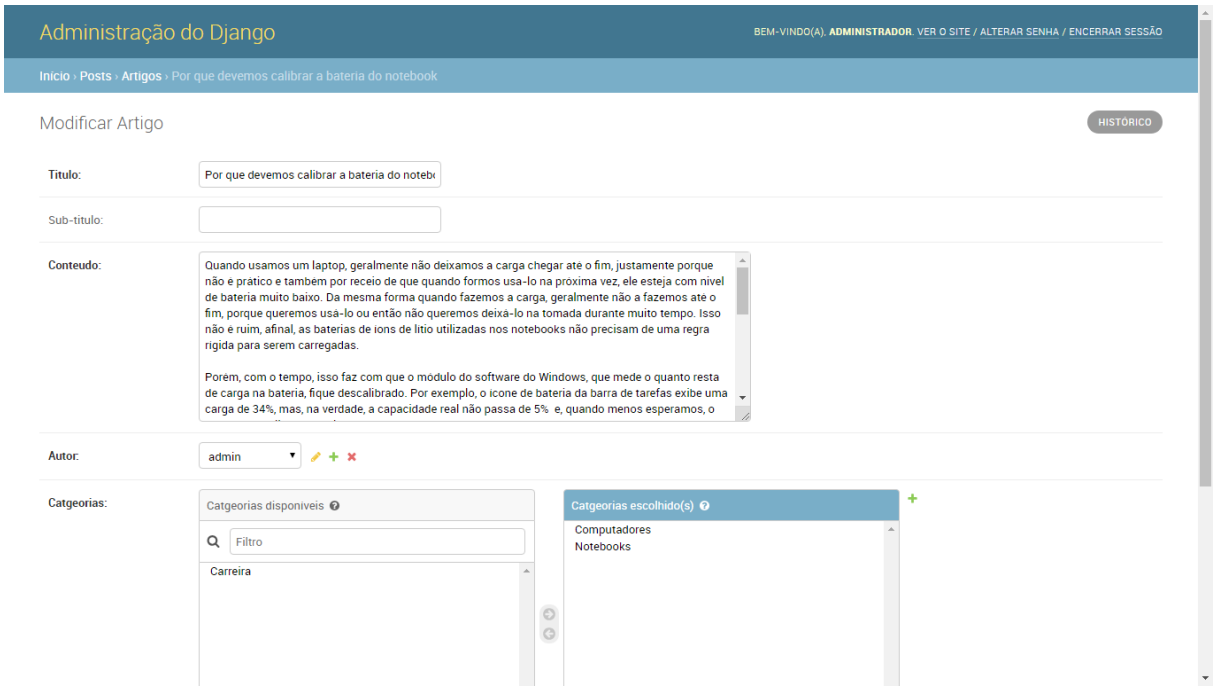
Fonte: elaborada pela autora.

Figura 13 – Página inicial do painel administrativo do Django



Fonte: elaborada pela autora.

Figura 14 – Página de modificação de artigos com Django – Parte 1



Fonte: elaborada pela autora.

Figura 15 – Página de modificação de artigos com Django – Parte 2

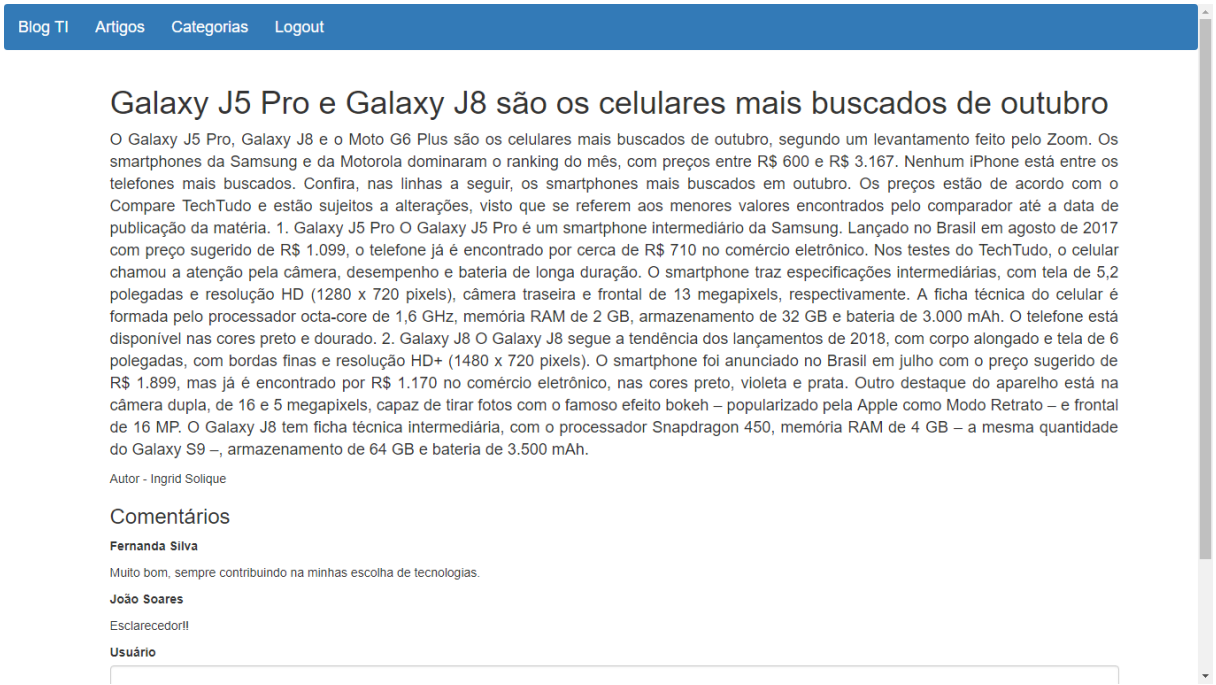
Fonte: elaborada pela autora.

O cenário descrito acima desenvolvido na tecnologia Meteor, gerou o sistema ilustrado pelas imagens a seguir.

Figura 16 – Página inicial do Blog com Meteor, apresenta a listagem dos artigos e botões para gerenciar os artigos

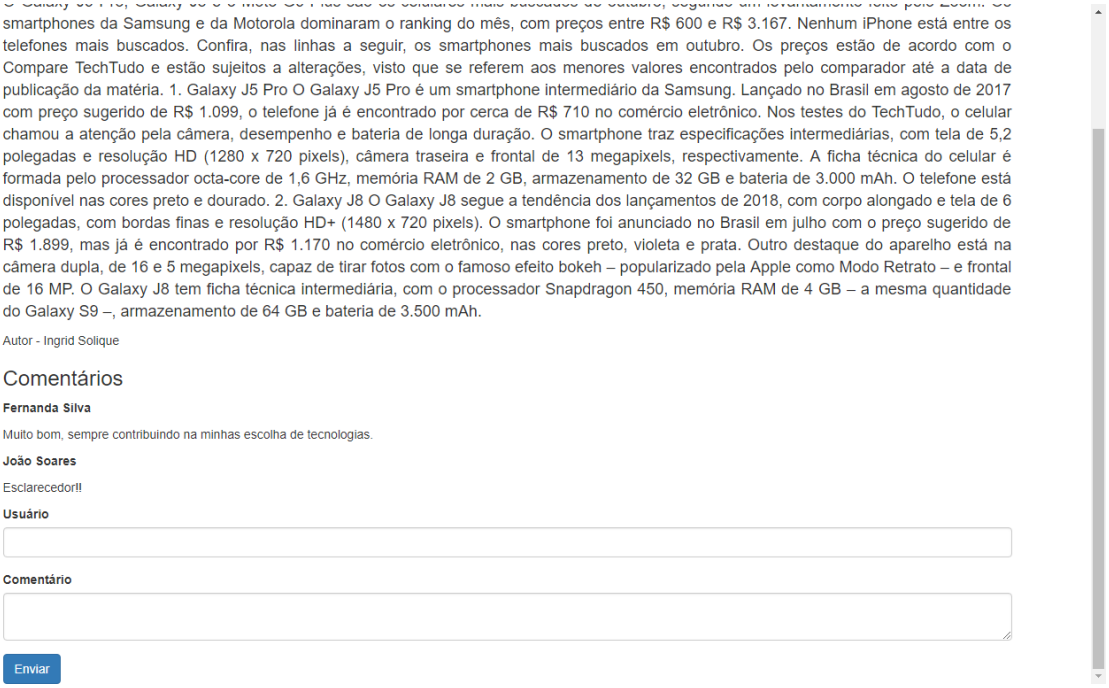
Fonte: elaborada pela autora.

Figura 17 - Página de visualização dos artigos com Meteor, apresenta todo o conteúdo do artigo e os comentários – Parte 1



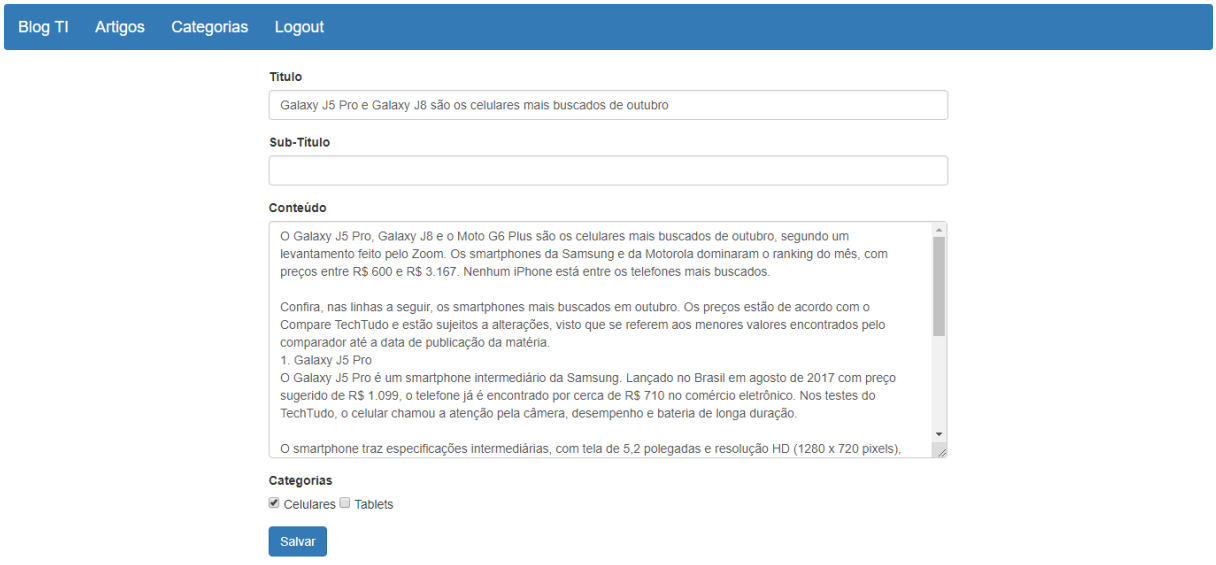
Fonte: elaborada pela autora.

Figura 18 - Página de visualização dos artigos com Meteor, apresenta todo o conteúdo do artigo e os comentários – Parte 2



Fonte: elaborada pela autora.

Figura 19 – Página de modificação do artigo com Meteor



Fonte: elaborada pela autora.