PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Graduação em Sistemas da Informação

Saulo Daniel Medeiros

PORTAL DA SAÚDE

Belo Horizonte
2018

SUMÁRIO

Lista de Figuras

1 INTRODUÇÃO

Nos dias atuais o uso de sistemas e aplicações tem aumentado significativamente, devido ao alto número de usuários com fácil acesso a computadores e smartphones, e também ao fácil acesso a internet. O uso dessas ferramentas auxiliam os usuários em suas atividades e se mostram úteis no dia a dia dos mesmos, podendo realizar atividades rotineiras e massantes de maneira automatizada deixando o usuário livre para realizar outras atividades. Essas ferramentas podem ser aplicadas em muitas áreas, inclusive no âmbito da saúde, auxiliando todos os profissionais e usuários que podem ter alguma ligação com esse segmento.

1.1 Motivação

O sistema de saúde brasileiro, visto que rege um país de grande porte, contempla uma grande diversidade de áreas e temas, possui algumas falhas das quais poderiam ser evitadas se houvesse uma melhor gestão dos recursos.

Segundo (NOGUEIRA; NEVES, 2008) dentro do Programa Saúde da Família(PSF), que tem como objetivo fazer uma gestão da saúde das famílias brasileiras segmentando equipes de saúde, contendo desde médicos e enfermeiros até agentes de saúde e colaboradores, alocadas em suas respectivas regiões e estados, possui um desperdício de informações e dados valiosos para o processo. Esse desperdício é gerado por diferentes fatores dentre eles o fato de que a maioria das informações coletadas sobre as famílias passam por uma hierarquia de processo, e grande parte dessas informações são transmitidas por meio de papel ou boca-a-boca, o que gera uma falha de comunicação dentro do processo. Essa falha de comunicação gera uma situação crítica, pois as informações e dados que deveriam chegar para os médicos e enfermeiros no momento de atendimento dos pacientes são incompletos ou inexistentes. Isso reflete diretamente na população, seja por falta de orientação ou pelo atendimento indevido ou incompleto, e pela sobrecarga dos profissionais da área da saúde em geral. Alguns desses problemas poderiam ser reduzidos ou até mitigados, caso houvesse um sistema ou aplicação para auxiliar nos assuntos de rotina e atendimento da população.

Como resultado desse trabalho espera-se possuir um projeto de aplicação web, acessível para usuários e profissionais da área da saúde, visando uma melhora durante o atendimento

dos pacientes das unidades de saúde, e fornecer informações úteis sobre a área da saúde para a população.

1.2 Objetivos

1.2.1 Geral

Criar um projeto de aplicação web com um foco em serviços de saúde e controle de vacinação da população, visando o auxilio para o atendimento dos pacientes dentro de unidades de saúde, e fornecer informações úteis para a população.

1.2.2 Específico

- Disponibilizar serviço de consulta da localização postos de saúde de Belo Horizonte e região metropolitana.
- Disponibilizar consulta online de cartão de vacinação por usuário.
- Disponibilizar consulta de dados do usuário para equipe médica autorizada.
- Criar um *feed* de notícias relacionadas com a área da saúde.
- Criar um mecanismo de notificação para o usuário sobre vacinas em atraso e futuras (1 mês).
- Disponibilizar consulta sobre hospitais de Belo Horizonte e suas respectivas especialidades, contendo uma breve descrição.

2 REFERENCIAL TEÓRICO

2.1 Aplicações Web

As aplicações Web possuem seu espaço dentro da área da computação, devido ao sucesso e crescimento do uso ao longo dos anos, e também pelo aumento de recursos disponíveis nesse contexto. Essas aplicações utilizam tecnologias e linguagens mundialmente conhecidas como HTML, JavaScript e CSS, que servem como base estrutural para o ambiente Web.

Segundo FRATERNALLI e PAOLINI (1998) pelo fato de utilizarem da infra-estrutura Web, as aplicações Web passam a possuir características específicas, e devem considerar particularidades relacionadas às dimensões:

- Estrutural (conceitual): define a organização das informações a serem tratadas pela aplicação e os seus relacionamentos.
- Navegacional: representa como as informações serão acessadas através da aplicação.
- Apresentação: descreve como as informações e o acesso a essas serão apresentados ao usuário da aplicação.

Essas dimensões podem definir diferentes visões para o projeto da aplicação Web.

”Processos de desenvolvimento para aplicações de software Web devem produzir representações para projeto de aspectos de aplicações tradicionais, como estrutura e funcionalidades; e também para aspectos orientados para Web, como navegação e apresentação (com recursos Web)”(MENDES; CONTE; TRAVASSOS, 2005). Existem diferentes tipos de metodologias voltadas para o desenvolvimento de aplicações Web, e a construção de modelos específicos se torna uma tarefa relevante, entretanto não se pode definir um modelo padrão.

2.2 Engenharia de Software

A Engenharia de Software é uma área que estuda os processos envolvidos no desenvolvimento de um sistema e quais as formas de otimizá-lo. Segundo SOMMERVILLE (2007) é uma área que estuda as possibilidades de melhorar a qualidade de um software, e um software

de qualidade é aquele que é fácil de usar, possui uma boa manutenibilidade, possui integridade dos dados, entre outras características relacionadas com a satisfação do usuário.

Segundo SOUZA (2015) a Engenharia de Software busca selecionar o método mais apropriado para para um conjunto de circunstâncias e uma abordagem mais criativa e menos formal pode ser mais eficaz em outras circunstâncias. Tendo em vista que, quando se trata de desenvolvimento, a qualidade e produtividade devem estar unidas para que o sistema se torne algo viável no meio empresarial, e para que isso seja possível os Engenheiros de Software tendem a procurar uma solução ágil para o desenvolvimento do produto.

2.3 Arquitetura de Software

Podemos dizer que arquitetura de software é um conjunto de elementos que definem a estrutura de um software através de seus componentes e relacionamentos, se preocupaando com a organização estrutural do problema, afim de solucioná-lo de maneira mais eficiente.

A Arquitetura de Software serve como uma ferramenta para estruturar de maneira gráfica a melhor solução para o desenvolvimento de um sistema, e este por sua vez podendo ser uma única aplicação ou se tratar de uma integração entre diferentes sistemas. Para que se possa alcançar o objetivo de montar uma boa estrutura, capaz de alcançar o resultado esperado, é necessário que haja uma visão crítica das possibilidades existentes, levando em consideração as vantagens e desvantagens de cada tipo de abordagem. Essa visão crítica sobre o problema ajuda durante todo o processo de estruturação da aplicação, sempre visando compreender o grau de complexidade das funcionalidades do sistema, para que problemas futuros sejam evitados.

2.4 Arquitetura MVC

A Arquitetura MVC (Model-View-Controller) faz uma abordagem para a estruturação de uma aplicação, separando em módulos ou camadas suas partes principais, possibilitando um sistema com uma alta coesão e um baixo acoplamento entre as classes. A coesão dentro da visão do paradigma O.O. (Orientada a Objetos), fala que para que uma classe possua uma alta coesão a mesma deve possuir responsabilidades e propósitos bem definidos. Já o acoplamento,

também dentro da visão O.O. fala sobre o grau de dependência entre as classes de um sistema, quanto maior a dependência de uma classe sobre os métodos de outra, maior será esse grau de acoplamento e vice-versa.

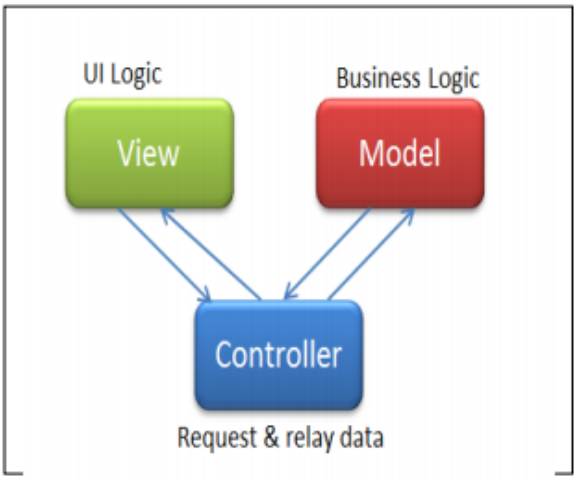
Segundo ASTOLFI (2012) o modelo MVC possui três camadas principais:

- Camada de Visão: é o estágio transitório entre a interface e o sistema. É a camada que interage com o usuário, permitindo que o mesmo obtenha acesso às informações geradas pelo sistema, e para isso estão presentes nessa camada as páginas, as funcionalidades e protocolos de transferência de dados.

- Camada de Serviço: também conhecida como camada de controle, é o estágio que são tratadas as regras de negócio da aplicação.É de responsabilidade dessa camada conhecer as entidades do sistema, e caso haja algum tipo de erro a camada de controle deve tratá-lo da maneira correta. É a responsável pela a comunicação entre a camada de visão e a camada de modelo.

- Camada de Modelo: É responsável por garantir a integridade dos dados, o modelo de relacionamento entre entidades e os atributos que compõem cada uma delas. É a base do sistema, é nessa camada que são declaradas todas as estruturas de dados, que pertencem ao sistema, permitindo então uma boa estruturação do código.

Figura 1 – Componentes MVC



Fonte: (SARKER; APU, 2014)

A figura 1 demonstra de uma maneira geral como é feita a comunicação entre as camadas da arquitetura MVC.

2.5 Aplicações Distribuídas

As aplicações distribuídas permitem que seus componentes estejam separados em vários ambientes de maneira heterogênea. Segundo TANENBAUM e STEEN (2008) podemos definir as aplicações distribuídas como uma coleção de computadores independentes que se comporta perante o usuário como um único sistema consistente.

A principal forma de comunicação de um sistema distribuído, é através da rede, possibilitando a troca de informações entre suas partes. Ao se desenvolver uma aplicação distribuída nos deparamos com alguns desafios que devem ser superados, para que tenhamos um resultado consistente e satisfatório, são eles :

- Heterogeneidade: diversos hosts devem se comunicar de maneira transparente, ou seja, várias instâncias da mesma aplicação devem ser capazes de se comunicar, independente da maneira como estão construídas.

- Abertura: devem possuir um ambiente que permita várias formas de integração, ou seja, deve prover um ambiente que possa ser estendido ou reimplementado, e para isso pressupõe a necessidade de criação de interfaces públicas, garantindo assim o acesso as funcionalidades, e uma padronização para facilitar a integração entre os componentes.

- Segurança: o ambiente deve fornecer segurança de acesso e proteção seletiva. Dentro dessa visão existem três aspectos fundamentais, a Confiabilidade que diz que somente o destinatário deve saber a mensagem enviada, a Integridade que diz que a mensagem deve chegar ao destinatário da mesma forma que foi enviada, ou seja não pode ter sido corrompida ou adulterada, e a Disponibilidade que diz que o serviço deve garantir o seu funcionamento mesmo diante de situações adversas.

- Escalabilidade: o sistema deve possuir alta escalabilidade, ou seja, permitir que seja possível a ampliação do sistema de acordo com a necessidade. Para isso deve ser implementado um controle entre os componentes do sistema, visando sempre evitar um gargalo para a aplicação.

- Manipulação de erros: erros do sistema devem ser tratados de maneira rápida e eficiente para minimizar os danos causados. Isso é possível através de um mecanismo de detecção

de falha, utilizando de técnicas já conhecidas para realizar sua tarefa como por exemplo a utilização de arquivos de log ou a replicação dos servidores.

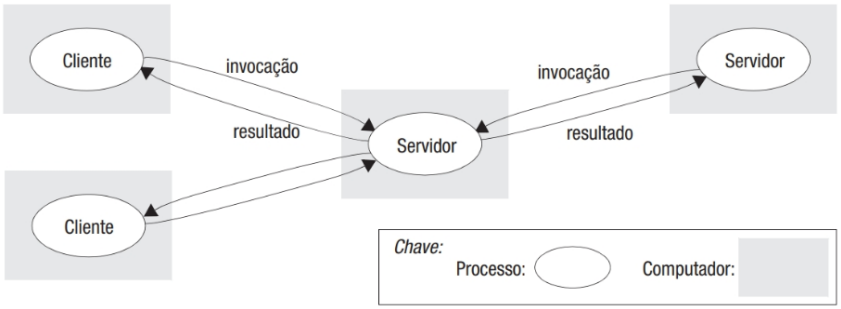
- Transparência: o sistema deve se comportar como se fosse uma aplicação local, transmitindo uma melhor experiência para o usuário. Dentro dessa visão temos alguns aspectos relevantes que são a Transparência de acesso que esconde as diferenças entre representação de dados e mecanismos de invocação, a Transparência de Localização que permite o acesso aos recursos sem que se conheça a localização física, a Transparência de Replicação que permite o uso de múltiplas instâncias de um recurso com o intuito de aumentar a confiabilidade e desempenho, mas sem que o desenvolvedor e o usuário saibam, a Transparência de Falhas que permite a conclusão de uma tarefa mesmo na ocorrência de uma falha, e a Transparência de Migração que permite a movimentação de recursos sem que sua operabilidade seja afetada.

Os middlewares são uma ou mais camadas de software que abstrai esses desafios das aplicações distribuídas, promovendo um ambiente integrado e coeso. Em outras palavras podemos dizer que os middlewares são programas que oferecem um ambiente intermediário entre a aplicação e os demais componentes do sistema.

Segundo COULOURIS (2013) a arquitetura de um sistema é sua estrutura em termos de componentes especificados separadamente. O modelo de arquitetura de um sistema distribuído primeiro simplifica e abstrai as funções dos componentes individuais do sistema e depois considera o posicionamento dos componentes em uma rede de computadores, buscando determinar padrões para a distribuição dos dados e da carga de trabalho, e considera também o inter-relacionamento entre os componentes, ou seja, seus padrões seus padrões funcionais e os padrões de comunicação. Para a construção de uma aplicação distribuída, existem diferentes modelos de arquitetura que podem ser utilizados, e devemos escolher o mais adequado para o objetivo de cada aplicação.

A arquitetura Cliente-Servidor é a mais utilizada quando se trata de sistemas distribuídos. Nesse modelo os processos clientes integram com os processos servidores, que podem estar em computadores distintos, e acessam recursos compartilhados. Os servidores podem também assumir o papel de cliente, visto que podem realizar requisições para outros servidores como pode ser visto na figura 2. Embora o modelo cliente-servidor ofereça uma estratégia direta e relativamente simples para o compartilhamento de dados e de outros recursos, ele não é flexível em termos de escalabilidade.

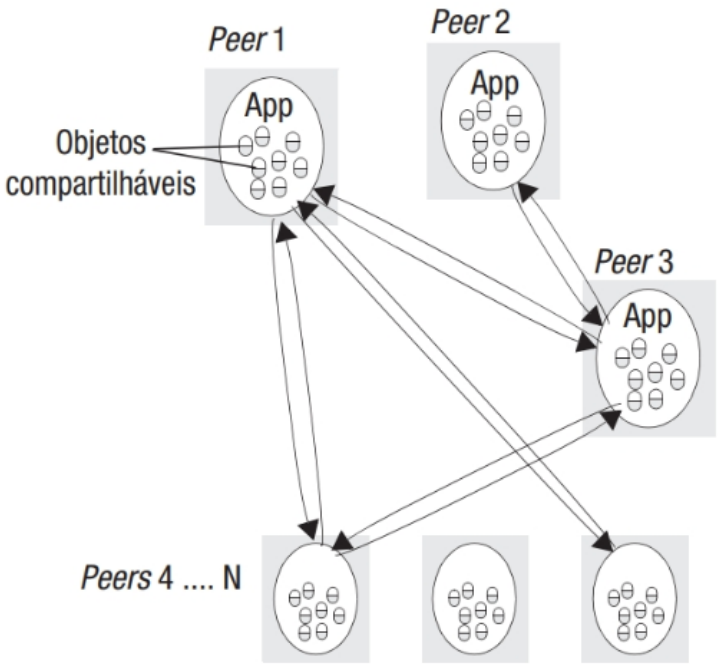
Figura 2 – Os clientes chamam o servidor individual



Fonte: (COULOURIS, 2013)

A arquitetura Peer-to-peer segundo COULOURIS (2013) considera que todos os processos participantes desempenham funções semelhantes, interagindo de maneira cooperativa como pares(peers) sema a distinção entre processos clientes e servidores, e nem entre computadores em que são executados. todos os processos executam o mesmo programa e oferecem o mesmo conjunto de interfaces entre si. A principal dessa arquitetura é utilizar a rede e os recursos computacionais dos usuários de um serviço para suportar o próprio serviço, ou seja, quanto maior o número de usuários mais recursos para serem utilizados. A figura 3 abaixo ilustra o formatado de uma aplicação utilizando a arquitetura peer-to-peer.

Figura 3 – Arquitetura Peer-To-Peer



Fonte: (COULOURIS, 2013)

Existem outros modelos de arquitetura que podem ser utilizados para a construção de

sistemas distribuídos, mas cada um haverá seus pontos positivos e negativos, cabendo aos desenvolvedores a tarefa de escolher aquele que melhor se adequa para a solução proposta.

2.6 JavaScript

O JavaScript é uma linguagem de programação interpretada, que foi criada com o intuito de ajudar os navegadores web a executar tarefas do lado cliente permitindo assim a interação com o usuário. Foi baseada na ECMAScript, outra linguagem de programação baseada em scripts. Algumas das características do JavaScript são:

- Interpretativa e estruturada: possue sintaxe de programação estruturada, com elementos de condicional que se asemelham a outras linguagens de programação, como por exemplo *if*, *switch* entre outros.
- Tipagem dinâmica: No JavaScript a tipagem é realizada através da atribuição de valores, permitindo que uma mesma variável possa ser de diferentes tipos, ou seja, a mesma variável pode ser um inteiro mas posteriormente se tornar em uma string.
- Baseada em protótipos: Essa linguagem utiliza um mecanismo de protótipos para tratar heranças entre classes, com isso é possível se assemelhar com as características da programação orientada a objetos.
- Orientada a eventos: no JavaScript a interação com o usuário é feita pelo HTML, através de eventos disparados pelos componentes da própria página web.

2.7 NoSQL - Not Only SQL

O NoSQL é uma abordagem de banco de dados diferente dos SGBDs convencionais que já são conhecidos por todos. Essa abordagem é capaz de armazenar diferentes tipos de estruturas. Segundo BONFIOLI (2006) os bancos de dados NoSQL são ótimos para trabalhar com grande volume de dados distribuídos.

Segundo LÓSCIO (2011) os bancos de dados NoSQL apresentam algumas características fundamentais que os diferenciam dos modelos de bancos de dados relacionais, permitindo que

armazenem um grande volume de dados não estruturados ou semi-estruturados. Essas características são:

- Escalabilidade horizontal: com o aumento do volume de dados a necessidade de aumento de desempenho relacionado a banco de dados é inevitável, e como solução para essa situação temos a escalabilidade horizontal onde threads/processos são criados para gerenciar e distribuir o processamento de dados. Nesse contexto um banco de dados relacional não se encaixaria muito bem, pois os diferentes processos realizando requisições sobre o mesmo conjunto de dados causaria uma alta concorrência o que prejudicaria o desempenho da aplicação. Os banco de dados NoSQL não possuem essas barreiras pelo fato de utilizarem técnicas que permitem a distribuição dos dados, sendo uma delas é o Sharding que consiste em dividir os dados em múltiplas tabelas a serem armazenadas ao longo de diversos nós de uma rede. A utilização dessa técnica transmite a complexidade de juntar os dados que foram separados para a aplicação, sendo necessário a utilização de joins para conseguir realizar uma consulta completa.

- Ausência de esquema ou esquema flexível: os bancos de dado NoSQL tem ausência completa ou quase total do esquema que define a estrutura dos dados modelados, facilitando em relação a escalabilidade e aumento da disponibilidade, porém não garante a integridade dos dados armazenados.

- Suporte nativo a replicação: o processo de replicação reduz o tempo gasto para recuperar as informações armazenadas. Existem dois tipos de abordagens para a replicação a *Master-Slave*(Mestre-Escravo) e a *Multi-Master*. Na abordagem *Master-Slave* cada escrita no banco resulta em N escritas no total, onde N é o número de nós escravos, e a escrita é realizada apenas no nó mestre que e refeita nos nós escravos pelo nó mestre. Já na abordagem *Multi-Master* assumimos que existem vários nós mestres, diminuindo o gargalo em casos de grande número de acessos em um conjunto de dados.

- API simples para acesso aos dados: os bancos de dados NoSQL possuem o foco em recuperação e acesso rápido e eficiente dos dados armazenados, e para que isso seja possível as APIs devem ser desenvolvidas para facilitar esse acesso, permitindo que qualquer tipo de aplicação seja capaz de acessá-la.

- Consistência eventual: pelo fato de que os dados são armazenados de maneira distribuída em diferentes nós de uma rede, a consistência desses dados nem sempre é mantida em

todos os nós. Isso se baseia em no teorema CAP (*Consistency,Availability e Partition tolerence*) onde fala que, em um determinado momento, só é possível garantir duas de três propriedades entre consistência, disponibilidade e tolerância à partição. Em contra partida as propriedades ACID não podem ser obedecidas simultaneamente. Tendo em vista esse ponto, a aplicação deve ser estruturada para tolerar inconsistência de dados com o intuito de garantir a disponibilidade.

2.8 MEAN Stack

3 METODOLOGIA

REFERÊNCIAS

ASTOLFI, B. A arquitetura mvc no desenvolvimento de jogos para navegadores. 2012.

BONFIOLI, G. F. Banco de dados relacional e objeto-relacional: Uma comparaÇÃo usando postgresql. 2006.

COULOURIS, G. e. a. *SISTEMAS DISTRIBUÍDOS Conceitos e Projeto*. 5. ed. Av. Jerônimo de Ornelas, 670,Santana ,Porto Alegre,RS: Bookman, 2013. ISBN 9780132143011.

FRATERNALLI, P.; PAOLINI, P. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications. 1998.

LÓSCIO, B. F. e. a. Nosql no desenvolvimento de aplicações web colaborativas. 2011.

MENDES, E.; CONTE, T.; TRAVASSOS, G. H. Processos de desenvolvimento para aplicações web: Uma revisão sistemática. 2005.

NOGUEIRA, G.; NEVES, J. Estratégia para a gestão da informação no programa saúde da família do governo brasileiro. 2008.

SARKER, I. H.; APU, K. Mvc architecture driven design and implementation of java framework for developing desktop application. 2014.

SOMMERVILLE, I. *Engenharia de software*. 8. ed. [S.l.]: Pearson Addison-Wesley, 2007.

SOUZA, F. P. e. a. Estudo de viabilidade do uso de arquitetura orientada a microsserviços para maximizar o reaproveitamento de código. 2015.

TANENBAUM, A.; STEEN, M. *Sistemas distribuídos: princípios e paradigmas.* 2. ed. [S.l.]: Pearson Prentice Hall, 2008. ISBN 9788576051428.