

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Bacharelado em Sistemas de Informação

Saulo Daniel Medeiros

SISTEMA DE CONTROLE DE VACINAÇÃO PESSOAL - CVPSIS

Belo Horizonte
2018

Saulo Daniel Medeiros

SISTEMA DE CONTROLE DE VACINAÇÃO PESSOAL - CVPSIS

apresentada ao programa de Curso de Bacharelado em Sistemas de Informação da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: João Caram Santos de Oliveira

Belo Horizonte
2018

RESUMO

Segundo a , o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (...) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão separadas entre si por ponto e finalizadas também por ponto.

Palavras-chave: latex. abntex. editoração de texto.

LISTA DE FIGURAS

FIGURA 1 – Os clientes chamam o servidor individual	12
FIGURA 2 – Arquitetura Peer-To-Peer	12
FIGURA 3 – Componentes MVC	15
FIGURA 4 – Servidor Node	19
FIGURA 5 – MEAN <i>Stack</i>	19
FIGURA 6 – Diagrama de Casos de Uso do CPVSIS	23
FIGURA 7 – Diagrama de Classes do CPVSIS	24
FIGURA 8 – Diagrama de Processo Consulta Cartão de Vacina	24
FIGURA 9 – Diagrama de Processo Cadastro de Vacinas	25
FIGURA 10 –Diagrama de Processo Consulta de Postos de Saúde	25
FIGURA 11 –Diagrama de Processo Cadastro de Dependentes	26
FIGURA 12 –Tela de Login	27
FIGURA 13 –Modal Esqueceu Senha	28
FIGURA 14 –Tela de Registro de Usuário 1	28
FIGURA 15 –Tela de Registro de Usuário 2	29
FIGURA 16 –Menu lateral	30
FIGURA 17 – <i>Form</i> de consulta de cartão de vacina	31
FIGURA 18 –Resultado de consulta de cartão de vacina	31
FIGURA 19 –Cadastro de Vacinas	32
FIGURA 20 –Lista de vacinas cadastradas	32

LISTA DE SIGLAS

ACID – Atomicidade, Consistência, Isolamento e Durabilidade

API – *Application Program Interface*

CAP – *Consistency, Availability e Partition tolerance*

CSS – *Cascading Style Sheets*

CVPSIS – *Sistema de Controle de Vacinação Pessoal*

HTML – *Hypertext Markup Language*

JSON – *JavaScript Object Notation*

MEAN – Mongo, Express, Angular e Node

MVC – *Model View Controller*

NoSQL – *Not Only SQL*

OO – Orientado a Objetos

SGBD – Sistema de Gerenciamento de Banco de Dados

SPA – *Single Page Application*

SUMÁRIO

1	INTRODUÇÃO	7
1.1	Motivação	7
1.2	Objetivos	8
1.2.1	Geral	8
1.2.2	Específico	8
2	REFERENCIAL TEÓRICO	9
2.1	Engenharia de Software	9
2.2	Arquitetura de Software	9
2.3	Aplicações Distribuídas	10
2.4	Aplicações Web	13
2.5	Arquitetura MVC	13
2.6	JavaScript	14
2.7	NoSQL - Not Only SQL	15
2.8	MEAN Stack	17
2.8.1	MongoDB	17
2.8.2	ExpressJS	18
2.8.3	Angular	18
2.8.4	NodeJS	18
2.8.5	O Stack	19
3	METODOLOGIA	20
3.1	Levantamento de Requisitos	21
3.1.1	Login	21
3.1.2	Cartão de Vacina	21
3.1.3	Dependentes	21
3.1.4	Postos de Saúde	22
3.1.5	Notícias	22
3.1.6	Consulta Paciente	22
3.2	Projeto de Sistema	22
3.2.1	Casos de Uso	22
3.2.2	Diagrama de Classes	23
3.2.3	Diagramas de Processo	24
4	IMPLEMENTAÇÃO	27
4.1	Tela de Login	27
4.2	Tela de Registro de Usuário	28
4.3	Tela principal	30
4.4	Tela Consulta de Cartão de Vacina	30
	REFERÊNCIAS	33

1 INTRODUÇÃO

Nos dias atuais a rotina corrida das pessoas tentando conciliar o tempo entre trabalho, família e na maioria das vezes estudos, faz com que não se tenha a devida atenção para alguns pontos importantes, dentre eles a saúde, mais especificamente o controle de vacinação. Enquanto estamos na infância os pais tendem a ser mais cautelosos quanto às vacinas a serem tomadas, e em qual data devem ser aplicadas, mas quando se chega na vida adulta a vacinação é deixada de lado. Segundo MORAES (2018) a infectologista Rosana Richtmann do Hospital e Maternidade Santa Joana, em São Paulo, relata que em relação as crianças o Brasil está muito bem com a vacinação, mas quando se trata de adultos o cenário é diferente pois existe uma lacuna entre o que é oferecido e o que é realmente feito.

A campanha de vacinação no Brasil não tem obtido os resultados esperados para o ano de 2018. No início deste ano o Sarampo, uma doença que já havia sido erradicada do país desde 2016, voltou a contaminar a população. Segundo G1 (2018) a vacinação contra o Sarampo foi abaixo do esperado em todo o país, e em 2017 21 estados não alcançaram a meta de imunizar 95% das crianças. Em 2018 foram registrados mais de mil casos de sarampo, sendo que a última vez que houve um registro de mesma proporção foi em 1999 com 908 casos confirmados. Tendo conhecimento sobre esses dados da campanha de vacinação, nota-se que há uma falha no controle de vacinação.

1.1 Motivação

O controle de vacinação nos dias de hoje é realizado através do cartão de vacinação, que é uma cartela com as informações sobre as vacinas tomadas pelo indivíduo, sua perda pode causar a inconsistência de informações pois na maioria das vezes a pessoa não se recorda das vacinas já administradas. No caso de perda do cartão de vacinas, as recomendações são procurar o posto de saúde no qual o indivíduo esta habituado a consultar e solicitar a segunda via, porém pode ocorrer de o histórico de vacinação não estar atualizado e não ser possível recuperar essas informações, sendo assim necessário colocar todas as vacinas em dia de acordo com a faixa etária (BRASIL, 2018).

Ainda sobre a perda do cartão de vacinas, caso o indivíduo não saiba qual a localização

do posto de saúde, a obtenção dessa informação não se encontra disponível de maneira fácil no site do governo, tornando a recuperação das informações sobre o cartão de vacinas um processo mais demorado.

Como resultado desse trabalho espera-se possuir uma aplicação web, para auxiliar no controle de vacinação pessoal.

1.2 Objetivos

1.2.1 Geral

Criar uma aplicação web com um foco em controle de vacinação pessoal, e fornecer informações úteis da área da saúde para o usuário.

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 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 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.

2.2 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 preocupando 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.

2.3 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.

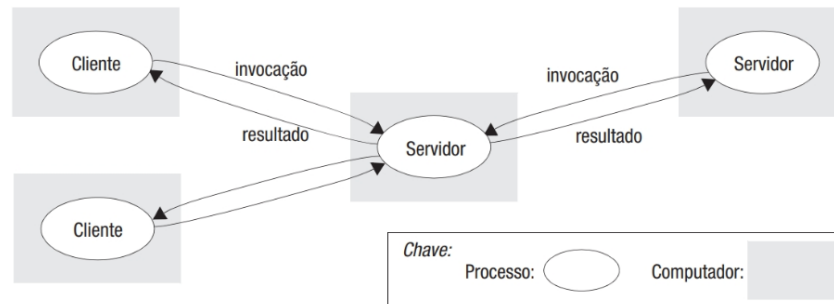
- **Transparência:** o sistema deve se comportar como se fosse uma aplicação local, transmitindo uma melhor experiência para o usuário. Dentro dessa visão temos alguns aspectos relevantes que são a Transparência de acesso que esconde as diferenças entre representação de dados e mecanismos de invocação, 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 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 1. 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.

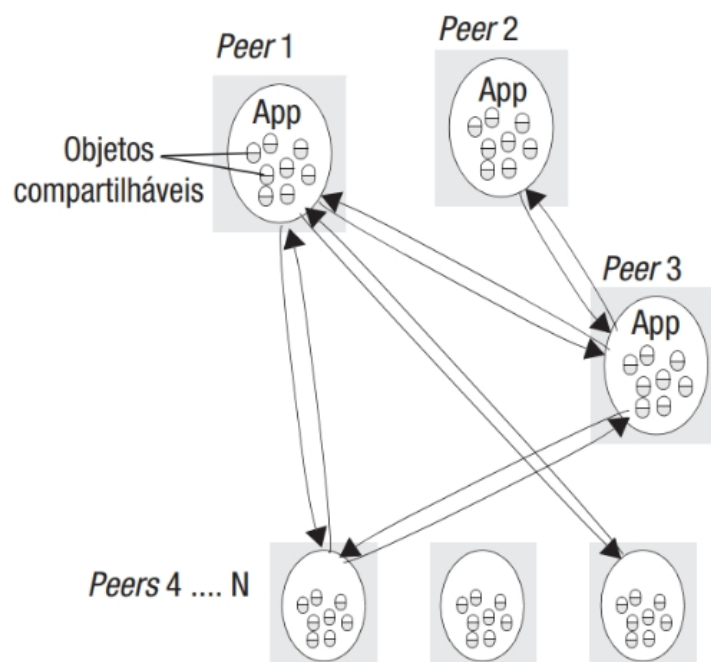
Figura 1 – Os clientes chamam o servidor individual



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) sem 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 2 abaixo ilustra o formato de uma aplicação utilizando a arquitetura peer-to-peer.

Figura 2 – 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.

2.4 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.

2.5 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.

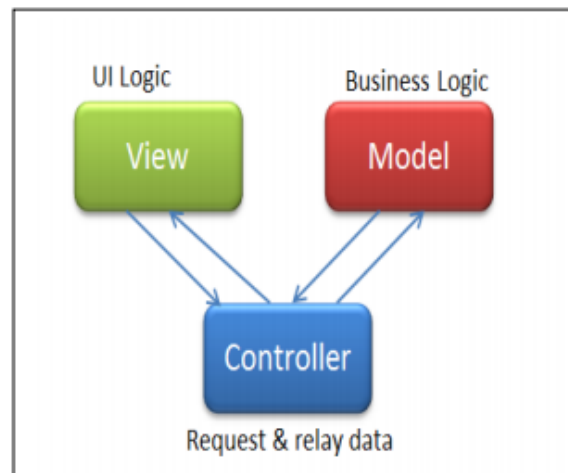
Segundo ASTOLFI (2012) o modelo MVC possui três camadas principais:

- 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 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.

A figura 3 demonstra de uma maneira geral como é feita a comunicação entre as camadas da arquitetura MVC.

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:

Figura 3 – Componentes MVC

Fonte: (SARKER; APU, 2014)

- Interpretativa e estruturada: possui sintaxe de programação estruturada, com elementos de condicional que se assemelham 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 tolerance*) onde fala que, em um determinado momento, só é possível garantir duas de três propriedades entre consistência, disponibilidade e tolerância à partição. Em contrapartida 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

O nome MEAN é um acrônimo das quatro tecnologias que compõe o MEAN Stack, MongoDB que é um banco de dados NoSQL, o framework backend ExpressJS, o framework frontend Angular e o servidor NodeJs.

2.8.1 MongoDB

O MongoDB é um banco de dados NoSQL de código aberto que é orientado a documentos. Esses documentos possuem o formato de JSON (*JavaScript Object Notation*) ou BSON que seria o JSON em formato binário, que são formatos muito utilizados para a troca de informações entre sistemas.

Segundo DB-Engines (2018) o MongoDB é o quinto banco de dados mais aceito pela comunidade de desenvolvimento, e como se trata de um banco de dados NoSQL possui um bom desempenho em relação a manipulação de grande volume de dados. Outros pontos fortes que justificam esse nível de aceitação são o fato de que trata diretamente com o formato JSON, o que facilita na configuração para comunicação backend da aplicação, e conseguir executar consultas em JavaScript.

2.8.2 *ExpressJS*

O ExpressJS é um framework backend que auxilia no desenvolvimento de aplicações em um servidor NodeJS, permitindo criar servidores web, utilizar requisições HTTP e também na criação e organização das rotas para os arquivos e diretórios da aplicação.

2.8.3 *Angular*

Segundo BONFIM e LIANG (2014) O Angular é um framework frontend que segue o modelo MVC e oferece um conjunto de ferramentas para a toda a criação e estilização de uma aplicação. Esse framework utiliza uma abordagem de SPA (*Single Page Application*), onde toda a aplicação é exibida em uma única página e seu conteúdo é carregado de acordo com as solicitações do usuário. Essa abordagem oferece um ganho no desempenho da aplicação, tendo em vista que não há a necessidade de recarregar toda a página a cada requisição feita, apenas partes da página são recarregados o que por sua vez diminui o tráfego de dados entre a aplicação e o servidor.

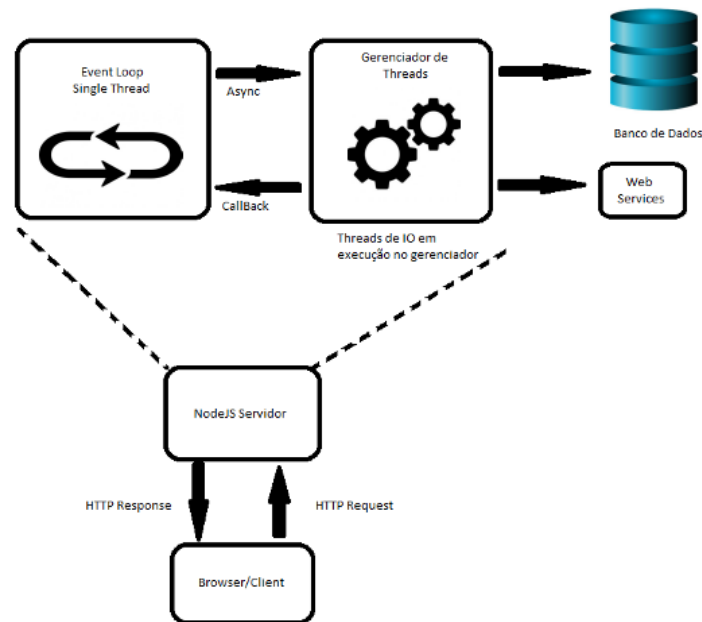
Outra abordagem utilizada pelo Angular é o *Two-Way Data Binding*, que facilita na exibição e manipulação dos dados carregados em tela, através de uma sincronização entre a tela e a camada de controle.

2.8.4 *NodeJS*

De acordo com PFÜTZENREUTER (2015) o NodeJS foi desenvolvido em 2009 com o intuito de executar programas Javascript fora do contexto do *browser*. A comunidade de desenvolvedores já almejava que o Javascript fosse uma linguagem de uso geral e o Node conseguiu atingir essa expectativa.

O Node para conseguir realizar essa tarefa de executar códigos Javascript fora do contexto do *browser*, adotou a filosofia assíncrona o que já é de costume com a execução dentro do *browser*. A figura 4 demonstra como é o funcionamento de um servidor Node.

Figura 4 – Servidor Node

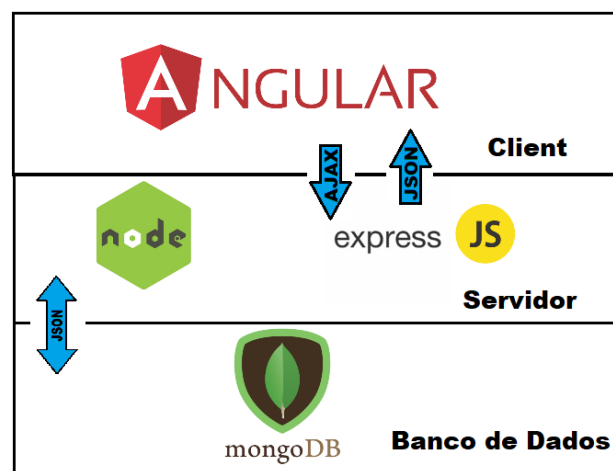


Fonte: Elaborado pelo autor

2.8.5 O Stack

O *MEAN Stack* utiliza essas tecnologias para criar toda a estrutura da aplicação, desde o lado *client* contendo o *layout* das páginas e as funcionalidades acessadas pelo usuário, até a camada de persistência de dados. A figura 5 mostra a estrutura do *MEAN Stack*.

Figura 5 – MEAN Stack



Fonte: Elaborado pelo autor

3 METODOLOGIA

O CVPSIS foi desenvolvido utilizando o modelo cascata com as seguintes etapas: levantamento de requisitos, criação do projeto e arquitetura do sistema, implementação. Essas etapas estão descritas a seguir.

O modelo cascata é um processo em sequencia onde a próxima etapa só é executada quando a etapa anterior é finalizada. Segundo SOMMERVILLE (2011) cada uma das etapas pode possuir documentos que serão utilizados na execução da próxima fase. O modelo consiste em 5 etapas:

- Definição de Requisitos;
- Projeto de Sistema;
- Implementação;
- Validação;
- Entrega e Manutenção;

Durante a execução dessas etapas segundo SOMMERVILLE (2011) podem ocorrer problemas nos resultados obtidos nas etapas anteriores, como problemas no levantamento de requisitos, problemas durante a implementação, o que pode levar a uma modificação do produto obtido entre as etapas, refletindo na execução da etapa seguinte.

Na etapa de Definição de Requisitos foram identificadas quais as funcionalidades que o CPVSIS deve ter para realizar o controle de vacinação pessoal. A partir da definição das funcionalidades foram gerados os requisitos funcionais para o sistema.

Na etapa de Projeto de Sistema, baseado nos requisitos levantados foi definido a arquitetura do sistema e as tecnologias a serem utilizadas.

Já na etapa de Implementação foi desenvolvido o sistema, com o objetivo de implementar os requisitos levantados na primeira etapa e seguindo a arquitetura definida na segunda etapa.

3.1 Levantamento de Requisitos

Segundo SOMMERVILLE (2011) os requisitos de um sistema são a descrição da tarefa a ser executada pelo sistema, e suas restrições funcionais. A partir dos requisitos deve-se ser capaz de executar as tarefas propostas.

Ao se utilizar técnicas de modelagem de processos é possível obter uma visão melhor do contexto onde o sistema será desenvolvido e como o mesmo deve funcionar, gerando assim a identificação de requisitos que reflitam as necessidades do negócio (BAKER, 2001). Este trabalho apresenta a modelagem de requisitos utilizando notação UML.

A seguir são apresentados os requisitos levantados nessa etapa.

3.1.1 *Login*

- O usuário deve logar no sistema para poder utilizá-lo.

3.1.2 *Cartão de Vacina*

- O usuário deve ser capaz de consultar o cartão de vacina cadastrado.
- O usuário deve ser capaz de consultar o cartão de vacina cadastrado de seus dependentes.
- O usuário deve ser capaz de gerenciar as vacinas suas próprias vacinas.
- O usuário deve ser capaz de gerenciar as vacinas de seus dependentes.

3.1.3 *Dependentes*

- O usuário deve ser capaz de gerenciar os seus dependentes.

3.1.4 Postos de Saúde

- O usuário deve ser capaz de consultar informações e localização dos postos de saúde de Belo Horizonte.

3.1.5 Notícias

- O usuário Administrador deve ser capaz de gerenciar o *feed* de notícias.

3.1.6 Consulta Paciente

- O usuário Médico deve ser capaz de consultar o cartão de vacinas de um Paciente autorizado.

3.2 Projeto de Sistema

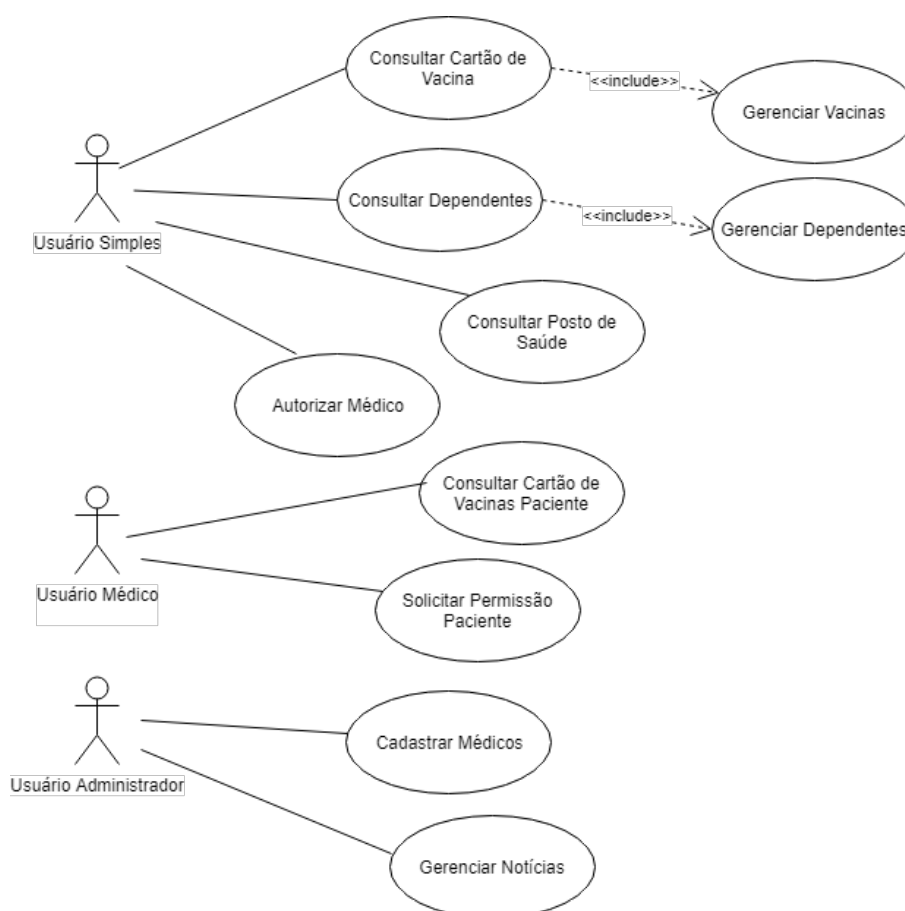
Segundo SOMMERVILLE (2011) um projeto de *software* deve descrever a estrutura do *software* a ser implementado, os tipos de dados utilizados pelo sistema, seus componentes e em alguns casos os algoritmos utilizados. A seguir serão apresentadas as estruturas utilizadas para a execução deste trabalho.

3.2.1 Casos de Uso

A figura 6 exhibe os principais casos de uso e os atores que interagem com o CVPSIS. Esse diagrama mostra os casos de uso de consulta e gestão do cartão de vacinas, consulta e gestão de dependentes, consulta de postos de saúde e autorização de médicos, sendo todos acessados pelo Usuário Simples. É exibido também os casos de uso de consulta do cartão de vacinas de paciente e solicitação de permissão acessados pelo Usuário Médico e os casos de

uso de cadastro de médico e gestão de notícias acessados pelo Usuário Administrador.

Figura 6 – Diagrama de Casos de Uso do CPVSIS

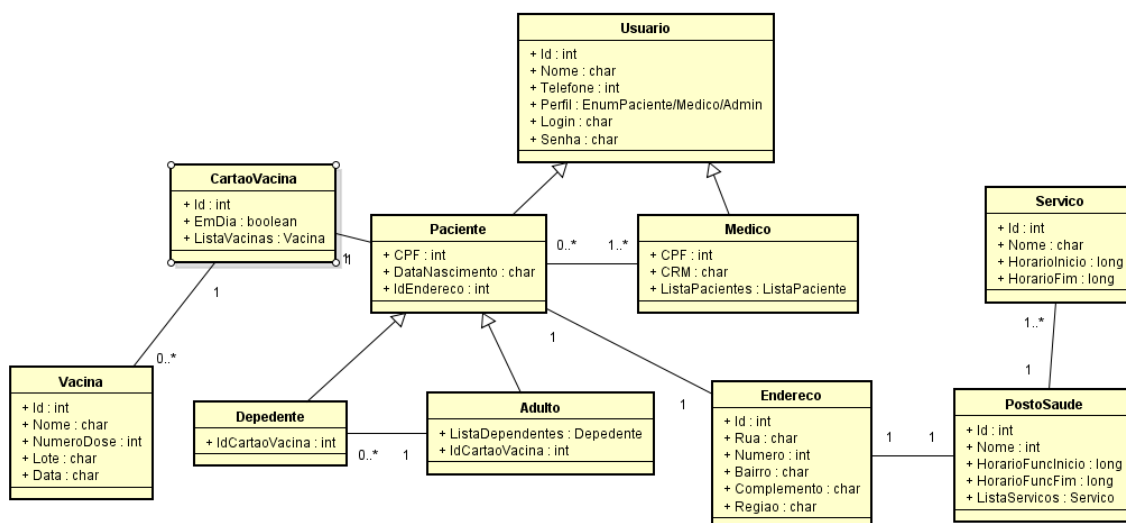


Fonte: Elaborado pelo autor

3.2.2 Diagrama de Classes

A figura 7 mostra as entidades utilizadas pelo sistema. A entidade Usuário representa o usuário que acessa o sistema, com seus dados de acesso. As entidades Paciente e Médico são especializações da classe usuário e cada uma possui seus atributos específicos. A entidade CartaoVacina representa o cartão de vacina de cada paciente que contém uma lista de vacinas representadas pela entidade Vacina. A entidade PostoSaude representa os postos de saúde que podem ser consultados dentro do sistema, que possui uma lista de serviços representado pela entidade Servico. A entidade Endereco é compartilhada entre a entidade PostoSaude e a entidade Paciente, pois ambos possuem um endereço associado.

Figura 7 – Diagrama de Classes do CPVSIS

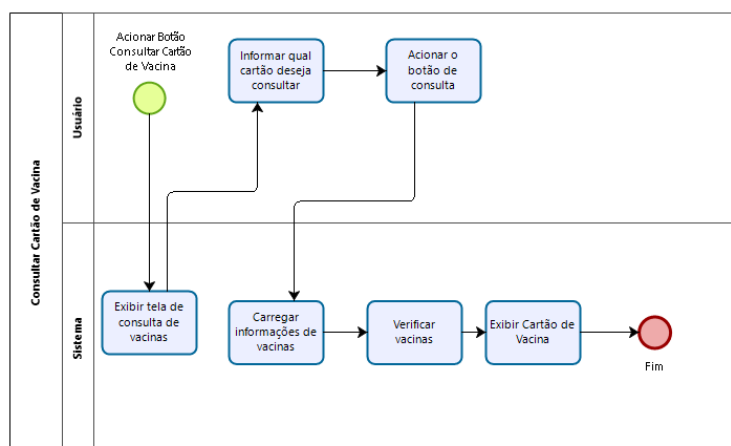


Fonte: Elaborado pelo autor

3.2.3 Diagramas de Processo

A figura 8 apresenta o fluxo executado pelo sistema no processo de consulta do cartão de vacinas do usuário. Ao acionar o botão de consulta de cartão de vacinas, o usuário é redirecionado para a tela de consulta. O usuário informa se deseja consultar o próprio cartão de vacinas ou se deseja consultar o cartão de um dependente, e então executa a consulta que retorna as vacinas já cadastradas.

Figura 8 – Diagrama de Processo Consulta Cartão de Vacina

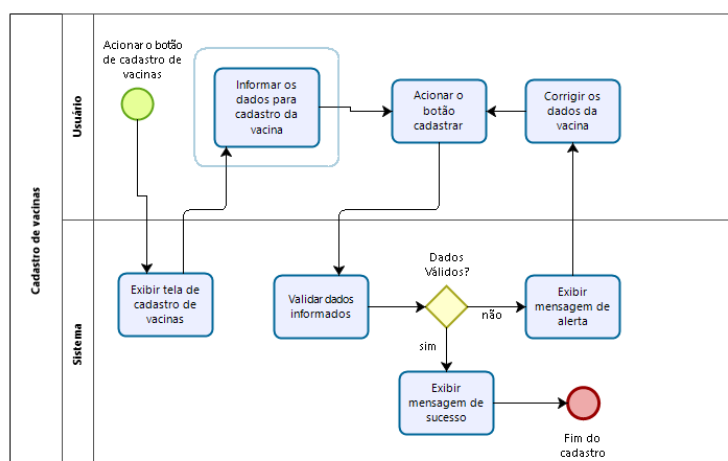


Fonte: Elaborado pelo autor

A figura 9 apresenta o fluxo executado pelo sistema no processo de cadastro de vacina.

Ao acionar o botão de cadastro de vacinas, o sistema exibe o formulário para preenchimento, após acionar o botão cadastrar o sistema valida os dados informados, caso estejam inválidos é exibido uma mensagem de alerta para o usuário, caso contrário é exibido a mensagem de sucesso do cadastro.

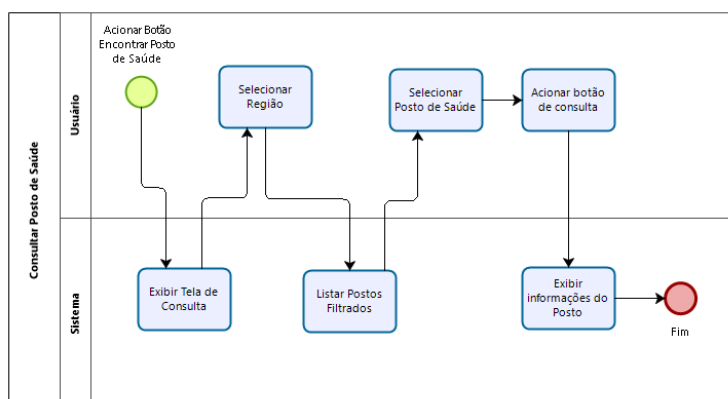
Figura 9 – Diagrama de Processo Cadastro de Vacinas



Fonte: Elaborado pelo autor

A figura 10 apresenta o fluxo de consulta de posto de saúde. Ao acionar o botão de consulta de postos de saúde o sistema exibe a tela de consulta, o usuário informa a região que deseja pesquisar, então o sistema carrega os postos de saúde referente a mesma. Ao selecionar o posto de saúde e acionar o botão de consulta, o sistema exibe as informações do mesmo.

Figura 10 – Diagrama de Processo Consulta de Postos de Saúde

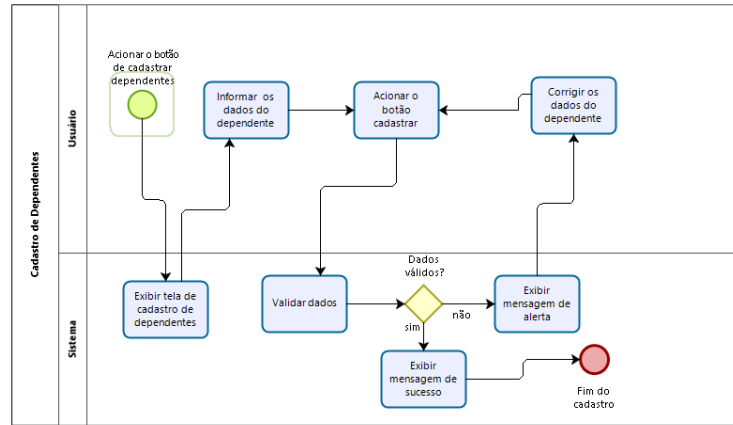


Fonte: Elaborado pelo autor

A figura 11 apresenta o fluxo executado pelo sistema no processo de cadastro de dependentes. Ao acionar o botão de cadastro de dependentes, o sistema exibe o formulário para preenchimento, após acionar o botão cadastrar o sistema valida os dados informados, caso es-

tejam inválidos é exibido uma mensagem de alerta para o usuário, caso contrário é exibido a mensagem de sucesso do cadastro.

Figura 11 – Diagrama de Processo Cadastro de Dependentes



Fonte: Elaborado pelo autor

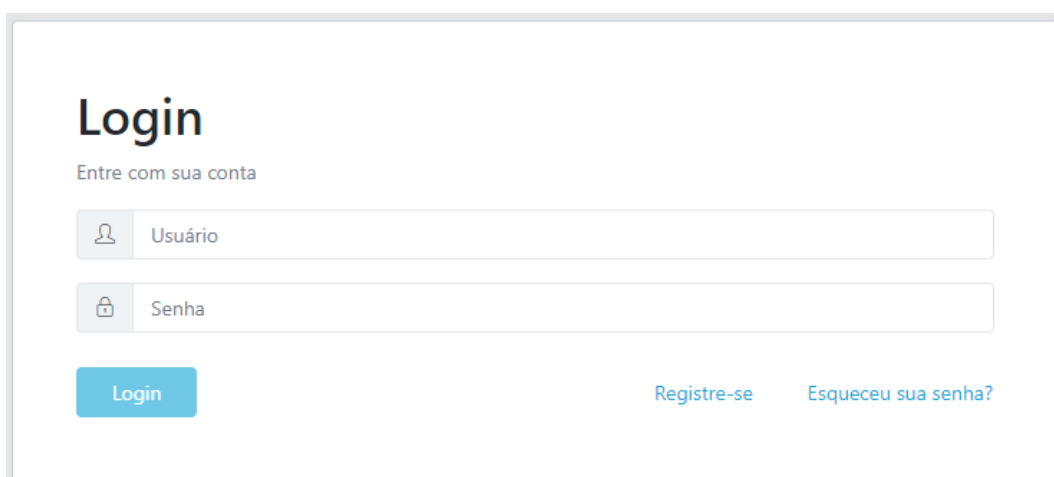
4 IMPLEMENTAÇÃO

Neste trabalho são apresentadas as telas do sistema e um descritivo de como foi elaborada cada interface.

4.1 Tela de Login

Quando o sistema é carregado, ou quando o usuário aciona o botão “Deslogar”, é apresentado a tela de login conforme a figura 12.

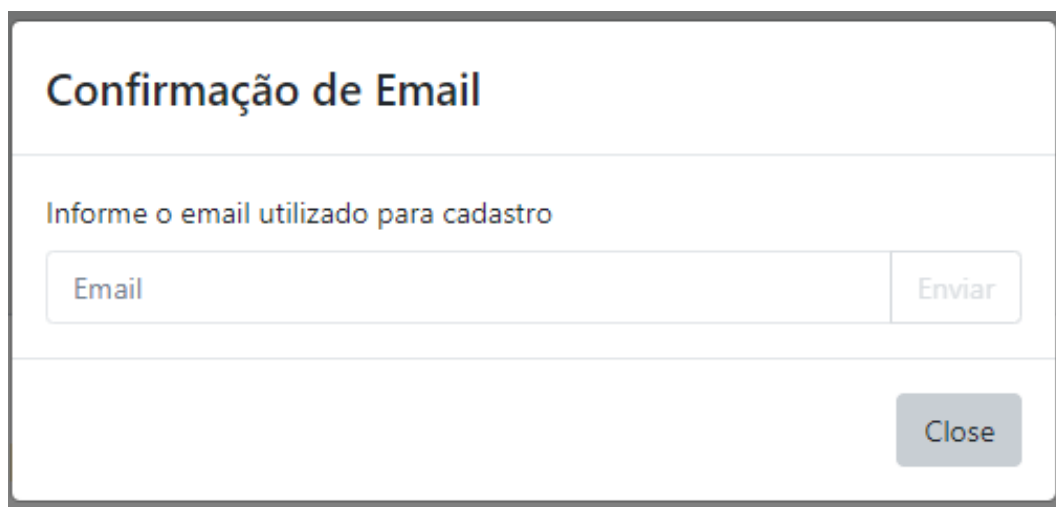
Figura 12 – Tela de Login



Fonte: Elaborado pelo autor

A tela de login é dividida em 3 etapas, seguindo as seguintes interações:

- O usuário informa login e senha cadastrados na base de dados do sistema e aciona o botão “Login”, é realizado a validação e redireciona para a tela principal do sistema. Caso seja informado login ou senha inválidos, é exibido um alerta no navegador informado o usuário que os dados estão incorretos.
- O usuário aciona o botão “Esqueceu sua senha?” e é exibido uma modal solicitando o e-mail cadastrado no sistema para envio de instruções de recuperação de senha conforme a figura 13.
- O usuário aciona o botão “Registrar-se” que redireciona para a tela de Registro de Usuário.

Figura 13 – Modal Esqueceu Senha

A screenshot of a modal window titled "Confirmação de Email". Inside the modal, there is a heading "Informe o email utilizado para cadastro". Below this, there is a text input field with the placeholder text "Email" and a button labeled "Enviar" to its right. At the bottom right of the modal, there is a button labeled "Close".

Fonte: Elaborado pelo autor

4.2 Tela de Registro de Usuário

Quando o usuário aciona o botão “Registrar-se” na tela de login é exibido a tela de registro de usuário conforme a figura 14

Figura 14 – Tela de Registro de Usuário 1

A screenshot of a registration form titled "Registrar-se". Below the title is the subtitle "Crie sua conta". The form contains four input fields, each with an icon on the left: a person icon for "Nome de usuário", an @ symbol for "Email", a lock icon for "Senha", and a lock icon for "Repita a senha". At the bottom of the form is a large green button labeled "Continuar".

Fonte: Elaborado pelo autor

A tela de registro de usuário é dividida em 2 etapas, executadas em ordem e com as seguintes interações:

- O usuário informa os dados solicitados pelo sistema, e aciona o botão “Continuar”, o sis-

tema valida se o nome de usuário informado está disponível e se a senha e a confirmação da senha estão iguais. Caso o nome de usuário esteja indisponível, é exibido um alerta informando o usuário que o nome de usuário já está sendo utilizado. Caso a senha e a confirmação da senha não estejam iguais, é exibido um alerta informando o usuário que a senha e a confirmação da mesma não conferem.

- O usuário informa os dados solicitados pelo sistema e aciona o botão “Registrar-se” conforme a figura 15. Caso algum campo não seja informado, o sistema exibe um alerta informando o usuário que um campo não foi preenchido. Quando o registro é completado é exibido um alerta informando o usuário que o registro foi realizado com sucesso e redireciona para a tela de login.

Figura 15 – Tela de Registro de Usuário 2



O formulário, intitulado "Informe seus Dados", é dividido em seções para coleta de informações pessoais e de endereço. A seção "Dados pessoais" contém campos para Nome, Sobrenome, CPF e Data de Nascimento (formato dd/mm/aaaa). A seção "Endereço" contém campos para Celular, Rua, Bairro, Número, Complemento e uma lista suspensa. Um botão verde "Registrar-se" está posicionado no final do formulário.

Informe seus Dados

Dados pessoais

Nome

Sobrenome

CPF

Data de Nascimento:

dd/mm/aaaa

Celular

Endereço

Rua

Bairro

Número

Complemento

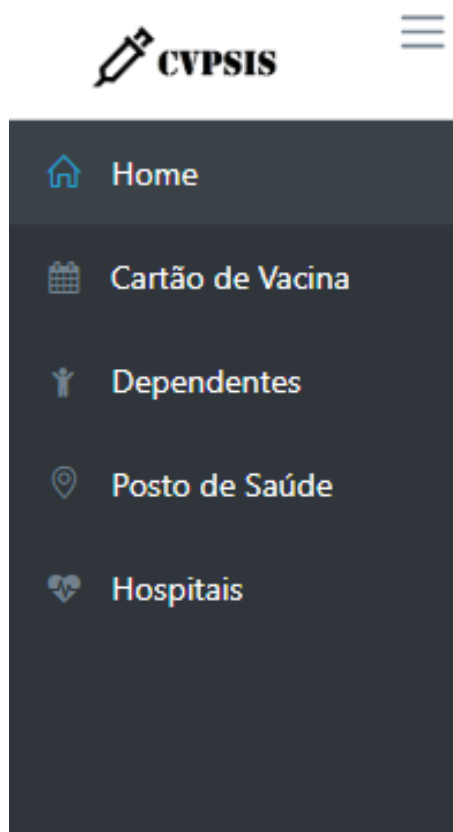
Registrar-se

Fonte: Elaborado pelo autor

4.3 Tela principal

Quando o usuário entra na tela principal é exibido o menu lateral que permite a navegação dentro do sistema de acordo com o perfil de usuário logado conforme mostra a figura 16.

Figura 16 – Menu lateral



Fonte: Elaborado pelo autor

Ao acionar um botão no menu o usuário é redirecionado para a tela respectiva da funcionalidade.

4.4 Tela Consulta de Cartão de Vacina

Ao acionar no menu lateral o botão “Cartão de Vacina” o usuário é redirecionado para a tela de consulta do cartão de vacina. A tela possui um form de consulta que permite o usuário consultar o próprio cartão de vacina ou o de seus dependentes conforme a figura 17.

O *form* de consulta possui 3 passos com as seguintes interações:

Figura 17 – Form de consulta de cartão de vacina

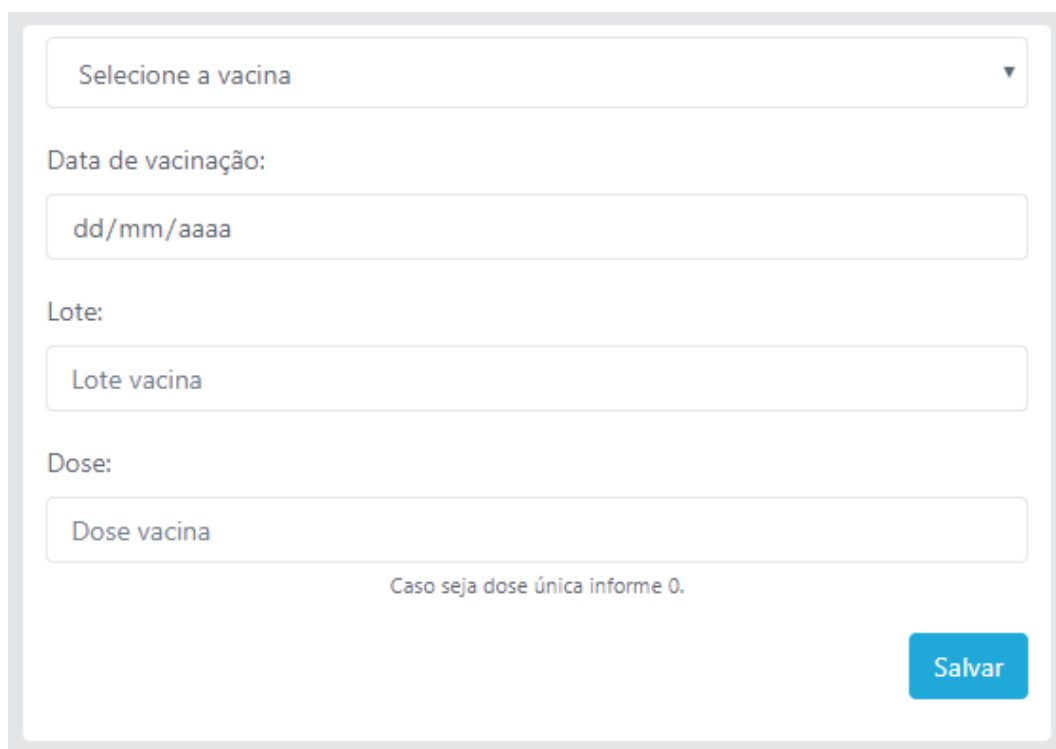
Fonte: Elaborado pelo autor

- O usuário seleciona o *checkbox* informando que deseja consultar o próprio cartão de vacina, e aciona o botão “Consultar”, então é apresentado o resultado com as vacinas do usuário conforme mostra a figura 18. Caso o usuário possua vacinas que não foram cadastradas, essas serão exibidas na cor vermelha.
- O usuário desmarca o *checkbox* e seleciona no combo informando que deseja consultar o cartão de vacina de um dependente e aciona o botão “Consultar”, então é apresentado o resultado com as vacinas do dependente. Caso o dependente possua vacinas que não foram cadastradas, essas serão exibidas na cor vermelha. Caso o usuário não possua dependentes cadastrados, o *checkbox* fica desabilitado sendo possível apenas a consulta de cartão de vacinas do próprio usuário.
- O usuário aciona o botão “Cadastrar” e é exibido o *form* para cadastrar uma vacina conforme a figura 19. O cadastro é de acordo com a seleção do *checkbox* ou do combo de dependentes. Caso o usuário já possua vacinas cadastradas é exibido uma lista permitindo a edição e exclusão das vacinas conforme a figura 20.

Figura 18 – Resultado de consulta de cartão de vacina

Nome Paciente: teste teste			
Nome: Antimeningocócica C conjugada Lote: ABC123 Dose: 1 Data: 01/10/2018	Nome: Antipneumocócica 10 valente conjugada Lote: Dose: Data:	Nome: BCG-ID Lote: Dose: Data:	Nome: DTP(tríplice bacteriana) Lote: Dose: Data:
Nome: Febre Amarela Lote: Dose: Data:	Nome: Hepatite B Lote: Dose: Data:	Nome: SRC (tríplice viral, MMR) Lote: Dose: Data:	Nome: Tetravalente (DTP + Hib) Lote: Dose: Data:
Nome: VOP(vacina oral contra a poliomelite - Sabin) Lote: Dose: Data:	Nome: VORH (vacina oral contra rotavirus humano) Lote: Dose: Data:		

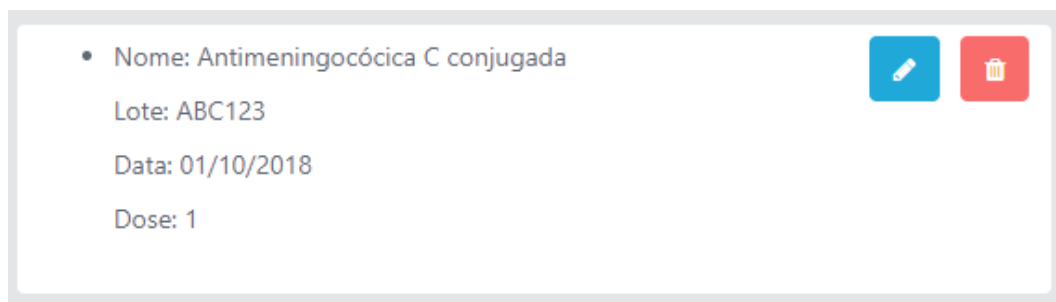
Fonte: Elaborado pelo autor

Figura 19 – Cadastro de Vacinas

The form is titled 'Cadastro de Vacinas' and contains the following fields and elements:

- A dropdown menu labeled 'Selecione a vacina' with a downward arrow.
- A label 'Data de vacinação:' followed by a text input field containing the placeholder 'dd/mm/aaaa'.
- A label 'Lote:' followed by a text input field containing the placeholder 'Lote vacina'.
- A label 'Dose:' followed by a text input field containing the placeholder 'Dose vacina'.
- A note below the dose field: 'Caso seja dose única informe 0.'
- A blue 'Salvar' button at the bottom right.

Fonte: Elaborado pelo autor

Figura 20 – Lista de vacinas cadastradas

The list displays one vaccine entry with the following details:

- Nome: Antimeningocócica C conjugada
- Lote: ABC123
- Data: 01/10/2018
- Dose: 1

At the top right of the entry are two icons: a blue square with a white pencil (edit) and a red square with a white trash can (delete).

Fonte: Elaborado pelo autor

REFERÊNCIAS

- ASTOLFI, B. A arquitetura mvc no desenvolvimento de jogos para navegadores. 2012.
- BAKER, B. Business modeling with uml: The light at the end of the tunnel. 2001.
- BONFIM, F. L.; LIANG, M. Aplicações escaláveis com mean stack. 2014.
- BONFIOLI, G. F. Banco de dados relacional e objeto-relacional: Uma comparação usando postgresql. 2006.
- BRASIL, G. do. *Postos de saúde disponibilizam vacinas gratuitas durante todo o ano*. 2018. Julho 6, 2018. Disponível em: <<http://www.brasil.gov.br/noticias/saude/2018/07/postos-de-saude-disponibilizam-vacinas-gratuitas-durante-todo-o-ano>>.
- COULOURIS, G. e. a. *SISTEMAS DISTRIBUÍDOS Conceitos e Projeto*. 5. ed. Av. Jerônimo de Ornelas, 670, Santana, Porto Alegre, RS: Bookman, 2013. ISBN 9780132143011.
- DB-ENGINES. *The DB-Engines Ranking ranks database management systems according to their popularity*. 2018. Setembro 15, 2018. Disponível em: <<https://db-engines.com/en/ranking>>.
- FRATERNALLI, P.; PAOLINI, P. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications. 1998.
- G1, S. *Estados enfrentam surto de sarampo, que volta a ameaçar o Brasil*. 2018. Julho 7, 2018. Disponível em: <<http://g1.globo.com/jornal-nacional/noticia/2018/07/estados-enfrentam-surto-de-sarampo-que-volta-ameacar-o-brasil.html>>.
- 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.
- MORAES, A. L. *Adultos também têm que tomar vacinas*. 2018. Julho 30, 2018. Disponível em: <<https://saude.abril.com.br/medicina/adultos-tambem-tem-que-tomar-vacinas/>>.
- PFÜTZENREUTER, E. *Node.js: ferramenta e filosofia*. 2015. Setembro 16, 2018. Disponível em: <<https://epxx.co/artigos/nodejs1.html>>.
- 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.
- SOMMERVILLE, I. *Engenharia de software*. 9. ed. [S.l.]: Pearson Education do Brasil Ltda, 2011. ISBN 9788579361081.
- 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.