Universidade de São Paulo

ESCOLA DE ENGENHARIA DE SÃO CARLOS

Departamento de Engenharia Elétrica e de Computação

Sistema de comparação de políticos com base na ideologia do usuário

Rafael Rodrigues Santana

São Carlos, SP

2019

Sistema de comparação de políticos com base na ideologia do usuário

Monografia final de conclusão do curso de engenharia elétrica com ênfase em eletrônica, apresentado ao Departamento de Engenharia Elétrica e de Computação — EESC - USP

Rafael Rodrigues Santana

Orientador: Prof. Marco Terra

São Carlos, SP

2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Santana, Rafael
S231s Sistema de comparação de políticos com base na ideologia do usuário / Rafael Santana; orientador Marco Terra. São Carlos, 2019.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2019.

1. Aplicação Web. 2. Modelo Evolucionário. 3. Cliente-Servidor. 4. MVC. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Rafael Rodrigues Santana

Título: "Sistema de comparação de políticos com base na ideologia do usuário"

Trabalho de Conclusão de Curso defendido e aprovado em 03/12/2019,

com NOTA (9,0), **pela Comissão Julgadora:**

Prof. Titular Marco Henrique Terra - Orientador - SEL/EESC/USP

Prof. Associado Evandro Luis Linhari Rodrigues - SEL/EESC/USP (docente aposentado)

Mestre Pedro Henrique Aquino Barra - Doutorando - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Este trabalho é dedicado a minha mãe Zenaide,
que sempre me apoiou e nunca deixou de acreditar em mim.

Agradecimentos

Primeiramente agradeço a todos os brasileiros, pois só tive a oportunidade de aprender os assuntos que eu amo em uma universidade tão incrível graças a vocês. Esse trabalho é uma forma de retribuir o que vocês fizeram por mim, e espero poder retribuir com muito mais.

Em segundo lugar, agradeço a todos os funcionários, professores e alunos, e a todas as pessoas que constroem a Universidade de São Paulo, proporcionando o aprendizado e a experiência que uma universidade de alto renome e respeito como a USP deve oferecer.

Em especial agradeço a todos os professores com os quais tive aulas e a todos os amigos que estiveram ao meu lado. Principalmente agradeço ao professor Marco Henrique Terra e ao professor Evandro Luis Linhari Rodrigues, que me auxiliaram no desenvolvimento desse trabalho, e aos meus amigos Mário Daniel e Gabriel Aranha, cujo apoio foi essencial para que o trabalho fosse desenvolvido.

Finalmente agradeço aos meus pais, que não só me incentivaram a ingressar na USP e a aprender sempre, como também possibilitaram isso das mais variadas formas, me ensinando a ler, escrever, e principalmente a ser uma pessoa íntegra, ao mesmo tempo que me possibilitaram poder estudar sem precisar ter outras preocupações além do meu aprendizado e formação.

Eu sonho poder contribuir para que um dia todas as crianças tenham as mesmas oportunidades que eu tive, pois tenho certeza de que a realização desse trabalho e a minha formação acadêmica e pessoal só foram possíveis graças a cada um de vocês.

Muito obrigado!

Resumo

Para que uma democracia seja real os políticos devem representar os cidadãos, principalmente ao votar nos projetos de lei. Para isso, é necessário que o cidadão preveja quais serão os votos dos candidatos caso sejam eleitos. Infelizmente, muitas vezes o candidato constrói em sua campanha eleitoral uma imagem que não condiz com suas atitudes, e assim acaba ludibriando os eleitores, que votam nele esperando um comportamento diferente do adotado por ele durante o mandato.

A melhor forma para se evitar isso é basear-se nos votos dos políticos em mandatos anteriores, pois com dados concretos sobre como os políticos já votaram, não é necessário confiar em promessas. O problema é que para se basear nesses votos, o cidadão deve descobrir todos os votos de todos os candidatos e os comparar aos seus próprios votos. Dada a dificuldade desse processo até hoje, que ainda não possui qualquer tipo de automação, grande parte dos eleitores votam com base principalmente na imagem que é construída sobre o político, tanto por ele mesmo quanto pela mídia.

Esse trabalho consiste no desenvolvimento de uma aplicação web que permite ao usuário encontrar qual ou quais políticos mais o representam, de forma automática, sem que o usuário precise sequer conhecer os nomes dos candidatos ao utilizar. Para isso, o usuário deve votar nos projetos mais importantes dos últimos anos. A partir desses dados é realizada uma comparação entre os votos do usuário e os votos dos políticos, levando em conta o grau de importância que o usuário dá a cada projeto. Então os políticos são ordenados em função da sua compatibilidade com o usuário.

Utilizando conceitos de Web 2.0 e 3.0, a metodologia de desenvolvimento conhecida como incrementação, e uma arquitetura cliente-servidor de 3 camadas modificada, a construção da aplicação foi planejada e executada, incluindo também elementos da arquitetura MVC. O resultado é uma aplicação web progressiva que pode ser executada tanto num sistema Android quanto em qualquer browser de navegação de internet, e que utiliza conceitos de Interface Humano Computador (IHC) para melhorar ao máximo a experiência e o entendimento do usuário por meio de um design simples e intuitivo.

Após disponibilizada a aplicação, ela foi testada por meio de pesquisas com os usuários, que responderam positivamente às perguntas, deixando claro que a aplicação cumpriu o objetivo de realizar uma comparação fácil, rápida, precisa e abrangente dos políticos atuais em relação às suas ideologias e crenças pessoais.

Palavras-Chave: Aplicação Web, Modelo Evolucionário, Cliente-Servidor, MVC.

Lista de Figuras

Figura 1 - Modelo em Cascata. 7
Figura 2 - Modelo Evolucionário. 9
Figura 3 - Modelo de Prototipagem Rápida. 10
Figura 4 - Modelo em Espiral. 12
Figura 5 - Comparação entre desenvolvimento baseado em plano e desenvolvimento ágil. 14
Figura 6 - Arquitetura cliente-servidor. 18
Figura 7 - Arquitetura cliente-servidor de duas camadas. 19
Figura 8 - Arquitetura cliente-servidor de três camadas. 20
Figura 9 - A arquitetura MVC. 22
Figura 10 - Metodologia de desenvolvimento. 25
Figura 11 - Divisão da aplicação na arquitetura Cliente-Servidor. 27
Figura 12 - Divisão da aplicação na arquitetura MVC. 28
Figura 13 - Fluxo de atividades geral. 29
Figura 14 - Fluxo de atividades para votar em projetos. 30
Figura 15 - Estrutura dos dados de um projeto. 31
Figura 16 - Tela de votos em projetos. 31
Figura 17 - Fluxo de atividades para comparar políticos. 32
Figura 18 - Comparação de políticos no servidor de aplicação. 32
Figura 19 - Estrutura dos dados de um político. 34
Figura 20 - Tela de visualização de políticos. 34
Figura 21 - Fluxo de atividades para criar conta. 35
Figura 22 - Estrutura dos dados de um usuário e alterações na estrutura dos dados de um projeto. 36
Figura 23 - Fluxo de atividades fazer login. 37
Figura 24 - Fluxo de atividades das interações entre usuários. 38
Figura 25 - Alterações nas estruturas dos dados de projetos e usuários. 39
Figura 26 - Estrutura final dos dados. 40
Figura 27 - Resultado da pergunta sobre a facilidade de utilização da aplicação. 41
Figura 28 - Resultado da pergunta sobre a abrangência da comparação da aplicação. 42
Figura 29 - Resultado da pergunta sobre a precisão da comparação da aplicação. 43
Figura 30 - Resultado da pergunta sobre o tempo de comparação dos políticos utilizando a aplicação. 44
Figura 31 - Resultado da pergunta sobre o tempo de comparação dos políticos sem utilizar a aplicação. 44

Lista de Tabelas

Tabela 1 - Comparação entre Web 1.0, Web 2.0 e Web 3.0. 6
Tabela 2 - Princípios dos métodos ágeis. 16
Tabela 3 - Comparação entre arquiteturas de duas e três camadas. 21

Lista de Abreviaturas e Siglas

WWW	*World Wide Web*
HTTP	*Hypertext Transfer Protocol*
HTML	*Hypertext Markup Language*
URI	*Uniform Resource Identifier*
AJAX	*Asynchronous Javascript and XML*
XML	*Extensible Markup Language*
RDF	*Resource Definition Framework*
SaaS	*Software as a Service*
MVC	Modelo-Visão-Controlador
PHP	*Hypertext Preprocessor*
CSS	*Cascading Style Sheets*
SQL	*Structured Query Language*
AQL	*Arango Query Language*
API	*Application Programming Interface*
IHC	Interação Humano-Computador

Sumário

1. Introdução 1
    1.1. Contextualização e Motivação 1
    1.2. Objetivo 2
    1.3. Organização do Trabalho 3
2. Fundamentos Teóricos 4
    2.1. Evolução Web 4
        2.1.1. Web 1.0 4
        2.1.2. Web 2.0 4
        2.1.3. Web 3.0 5
        2.1.4. Web 4.0 5
        2.1.5. Considerações Finais 6
    2.2. Metodologia de Desenvolvimento 7
        2.2.1. Modelo em Cascata 7
        2.2.2. Modelo Evolucionário 9
        2.2.3. Modelo de Prototipagem Rápida 10
        2.2.4. Modelo em Espiral 12
        2.2.5. Métodos Ágeis 14
        2.2.6. Considerações Finais 17
    2.3. Arquiteturas 18
        2.3.1. Arquiteturas Cliente-Servidor 18
            2.3.1.1. Arquitetura Cliente-Servidor de Duas Camadas 19
            2.3.1.2. Arquitetura Cliente-Servidor de Três Camadas 20
        2.3.2. Arquitetura Modelo-Visão-Controlador (MVC) 22
        2.3.3. Considerações Finais 23
3. Desenvolvimento 24
    3.1. Definição de Requisitos 25
    3.2. Planejamento de Arquitetura e Ferramentas 26
    3.3. Implementação 29
        3.3.1. Comparação de Políticos 30
            3.3.1.1. Votar em Projetos 30
            3.3.1.2. Comparar Políticos 32
        3.3.2. Armazenamento de Dados dos Usuários 35
            3.3.2.1. Criar Conta 35
            3.3.2.2. Fazer Login 37
        3.3.3. Interações entre Usuários 38
    3.4. Estrutura Final dos Dados 40
4. Testes e Resultados 41
5. Conclusão 45
6. Referências 46

1. Introdução

1.1. Contextualização e Motivação

A democracia teve início em 594 a.C., na cidade de Atenas, na Grécia, quando Sólon iniciou a reforma política que mudou a forma como o poder era formado. Até então, a governança da Grécia era passada de forma exclusivamente hereditária. A partir dessa reforma, os cargos mais altos do governo passaram a ser ocupados pelos cidadãos mais ricos. Como teoricamente qualquer cidadão poderia enriquecer, qualquer cidadão poderia vir a governar a Grécia, independente do seu nascimento, configurando assim um importante passo em direção à democracia que temos hoje. Além disso, a reforma política de Sólon instituiu as primeiras assembléias populares, chamadas de eclésias, onde pela primeira vez as decisões relativas a Atenas eram tomadas em consenso pelos cidadãos atenienses. (Encyclopaedia Britannica, Inc.)

Inicialmente, a definição de cidadão excluía as mulheres e os escravos. Com o passar do tempo isso mudou, e hoje todas as pessoas acima de uma certa idade podem votar. Além disso, duas diferentes formas de democracia passaram a ser adotadas: a democracia direta, onde cada cidadão vota diretamente em cada decisão a ser tomada, e a democracia indireta ou representativa (adotada no Brasil), onde cada cidadão vota em um grupo de pessoas para tomar essas decisões por elas.

Infelizmente, a democracia representativa apresenta um grande problema: a dificuldade em se encontrar o candidato mais adequado. Isso faz com que muitas vezes as pessoas votem nos candidatos baseando-se em fatores que podem levar ao erro, como promessas eleitorais e propagandas na mídia.

Hoje há ferramentas digitais que se propõem a auxiliar o eleitor nessa hora, mas na maioria dos casos elas nada mais são que o reflexo da opinião do autor, resultando em uma lista fixa de políticos recomendados para qualquer usuário, com base em características escolhidas e pontuadas exclusivamente pelo autor (ex: rankingpoliticos.org). Até mesmo as ferramentas mais avançadas criadas para esse fim, que levam em consideração as ideologias e crenças dos usuários como um todo, deixam a desejar, pois dão como resultado o mesmo político ideal do grupo para qualquer eleitor, independente de suas crenças políticas (ex: votenaweb.com).

Para que a democracia funcione, é necessário que os cidadãos votem nos candidatos que realmente os representam e lutam por eles. E para que eles façam isso, eles precisam primeiro saber quem são esses candidatos, tomando como base não as ideologias e crenças políticas de terceiros, mas sim as suas próprias opiniões, individualmente. Foi daí que surgiu a motivação para a realização desse trabalho.

1.2. Objetivo

O objetivo desse trabalho é auxiliar as pessoas a comparar todos os candidatos em que podem votar, tomando como base as suas ações dentro do congresso nos mandatos atuais e anteriores, de forma que essa comparação seja mais fácil, rápida, precisa e abrangente do que os métodos atuais. Essa comparação deve visar encontrar o político que mais se aproxima da pessoa que está utilizando o sistema, em termos de crenças políticas e de pautas defendidas.

Como cada pessoa tem ideologias diferentes e defende pautas diferentes, é necessário que o sistema desenvolvido para cumprir esse objetivo apresente um resultado diferente para cada pessoa, de acordo com a sua personalidade. Para isso, é necessária uma forma direta e automática de realizar essa comparação, onde cada usuário entra com dados que reflitam as suas crenças políticas e as pautas que defendem, e tenha como resultado o político que mais o representa no congresso.

Para maximizar a precisão dessa comparação, é essencial que os dados dos políticos e dos usuários sejam da mesma natureza, e também que estejam diretamente ligados a realidade, principalmente por parte dos políticos. Caso contrário, eles podem fornecer dados fictícios pensando apenas em conseguir o maior número de eleitores, e não correspondendo a isso durante seus mandatos. Além disso, os dados devem ser altamente relevantes.

O ideal seria utilizar como base os votos dos políticos no congresso, que são efetivamente transformados em leis, já que é nessa etapa do processo democrático que ocorre a transformação da vontade popular nas regras vigentes, o que indica a alta relevância desses dados. Além disso, esses votos são fatos passados e imutáveis, o que impossibilita a falsificação desses dados por parte dos políticos para conseguir eleitores. Felizmente, o portal da transparência nos disponibiliza esses dados.

Com os dados dos votos dos políticos no congresso em mãos, é necessário coletar dados da mesma natureza dos usuários. Para isso, basta perguntar aos usuários quais seriam os seus votos nesses mesmos projetos. Utilizando esses votos, tantos dos usuários quanto dos políticos, torna-se possível uma comparação fácil, rápida, precisa e abrangente, que retorna a cada usuário o político específico que efetivamente mais o representa no congresso.

Além da comparação de votos entre candidato e usuário, o sistema também irá disponibilizar informações sobre os políticos, que compreendem desde dados básicos como idade, naturalidade, cargo, etc. até informações extremamente importantes para se identificar a qualidade do trabalho de um político, como a lista dos projetos escritos ou votados por ele. Além disso, funções complementares também serão disponibilizadas, como definição de pesos para projetos de importância diferentes para o usuário e comunicação entre usuários para que discussões aprofundadas sobre os projetos possam acontecer, expondo assim pontos de vista diferentes.

1.3. Organização do Trabalho

O trabalho está dividido em cinco capítulos.

No primeiro capítulo foi apresentado um panorama geral dos assuntos que serão abordados no trabalho.

O segundo capítulo apresenta toda fundamentação teórica empregada no trabalho, desde as considerações relevantes e os princípios de desenvolvimento web e suas limitações, bem como as bases conceituais em cima das quais o mundo digital é transmitido e apresentado a nós através da web, como arquiteturas e metodologias de desenvolvimento utilizadas.

O terceiro capítulo tem a apresentação das ferramentas e métodos empregados no trabalho, detalhando os diagramas de fluxo de atividades e linguagens de programação utilizadas ao longo da parte prática do desenvolvimento.

O quarto capítulo contém os resultados do trabalho, apresentando uma discussão dos testes realizados e resultados obtidos.

E por fim, o quinto capítulo fornece a conclusão do trabalho com base nos resultados obtidos, além de análises gerais em torno do objetivo inicial e metodologias, arquiteturas e ferramentas utilizadas.

2. Fundamentos Teóricos

2.1. Evolução Web

Atualmente, web é o termo mais utilizado para se referir à *World Wide Web* (WWW), ou rede mundial de computadores. Ela compreende um sistema de documentos que são executados na internet. Nesse capítulo será discutida a história da evolução da Web, com base nos trabalhos de Aghaei et al (2012) e Choudhury (2014).

2.1.1. Web 1.0

A web 1.0 foi a primeira geração da web, com início em 1996, como informa a Tabela 1. Era majoritariamente apenas de leitura (*read-only*), composta por páginas estáticas que usavam linguagem de marcação básica e sua principal função para organizações era marcar sua presença online e disponibilizar suas informações para qualquer pessoa que quisesse conhecê-las melhor ou entrar em contato.

Suas principais limitações eram a falta de conteúdo compatível com máquinas (podia ser compreendido apenas por seres humanos) e o fato de o *web master* ser o único responsável por atualizar e gerenciar o conteúdo. Os principais protocolos da web 1.0 eram HTTP (*Hypertext Transfer Protocol*), HTML (*Hypertext Markup Language*) e URI (*Uniform Resource Identifier)*.

2.1.2. Web 2.0

A segunda geração da web teve início em 2004, como informa a Tabela 1, e ficou conhecida como web 2.0. Era de escrita e leitura (*read-write*) e por isso possibilitou avanços importantes como práticas colaborativas e maior participação do usuário na interação e no desenvolvimento do conteúdo web.

Nesse contexto, a web se tornou uma plataforma com software acima do nível de um único dispositivo. Essa interação aumentada do usuário mudou tanto a arquitetura de negócios, com o advento das lojas online, quanto as relações sociais entre as pessoas, por meio das redes sociais, blogs e qualquer aplicação que facilite a troca e produção coletiva de conhecimento.

Suas principais limitações eram os ciclos de mudanças e atualizações constantes, a falta de interconectividade entre diferentes plataformas e questões éticas relacionadas à construção e uso da

web 2.0. As principais ferramentas da web 2.0 eram AJAX (*Asynchronous Javascript and XML*), Adobe Flex e Google Web Toolkit.

2.1.3. Web 3.0

A web 3.0, ou web semântica, teve início em 2016 e é o estado atual da web, buscando diminuir as tarefas e decisões humanas e transferi-las para as máquinas, utilizando conteúdo web que pode ser lido por computadores. Isso também possibilita o compartilhamento e re-utilização dos dados por diversas aplicações, organizações e comunidades diferentes.

Para que isso aconteça é necessário que os dados estejam semanticamente estruturados de forma a possibilitar sua integração. Tim Berners-Lee, o inventor da World Wide Web, introduziu em 2017 um conjunto de regras de publicação conhecido como princípios de dados ligados (*linked data principles*) para conectar os dados da web. As regras são: Utilizar URIs como nomes para as coisas; Utilizar URIs HTTP para procurar por esses nomes; Incluir informações úteis para as máquinas utilizando o padrão RDF (*Resource Definition Framework*); Incluir links para outras URIs para associar informações relacionadas.

As maiores características presentes na Web 3.0 são o modelo de negócios SaaS (*Software as a Service*), plataformas de software de código aberto, bancos de dados distribuídos, "*resource pooling*" (onde os recursos são agrupados e servidos a múltiplos clientes de forma escalável) e web inteligente. Seus maiores desafios são a vastidão da web, que acarreta redundância nos dados, a inconsistência dos dados e também usuários mal-intencionados, que procuram enganar e se aproveitar desonestamente de outros usuários.

2.1.4. Web 4.0

A web 4.0, também conhecida como web simbiótica, é um ambiente em que a mente humana e as máquinas podem trabalhar e interagir em simbiose. Ainda é um conceito em desenvolvimento e não há uma definição exata de como ela será, mas em termos simples as máquinas serão inteligentes o suficiente para ler o conteúdo da web e decidir o que executar primeiro para carregar os websites mais rápidos e de qualidade superior. Será a web de leitura-escrita-execução-simultaneidade. Não há uma idéia exata de quais ferramentas serão utilizadas, mas está claro que a web está se movendo na direção de utilizar inteligência artificial para se tornar o mais inteligente possível e funcionar em simbiose com a mente humana.

2.1.5. Considerações Finais

A evolução da web é um processo natural que resulta dos avanços tecnológicos combinados com o aumento da capacidade de acesso da população à rede mundial de computadores. É interessante notar que as características entre as fases da web não são excludentes. Por exemplo, é possível utilizar conceitos da web 2.0 e da web 3.0 ao mesmo tempo, como foi feito nesse trabalho.

O resultado é uma aplicação com características de web social (interação entre usuários), mas também com características da web semântica, tanto em relação ao significado dos dados quanto à apresentação e visualização dos mesmos. Como a maioria dos dados apresentados vem de um banco de dados com vários dados da mesma classe para políticos ou projetos diferentes, utiliza-se as classes desses dados para buscá-los no banco, estabelecendo-se assim uma semântica para organização e manutenção do código a longo prazo. E como o próprio *framework* utilizado traz funções próprias de apresentação que já vem classificadas com nomes específicos, cada elemento que aparece na tela é nomeado de acordo com sua função na apresentação.

Além disso, os usuários não só podem ler e escrever conteúdo como também podem executar a aplicação ao buscar por um político compatível. Finalmente, o próprio desenvolvimento desse trabalho se enquadra na característica da web 3.0 em que as pessoas constroem aplicações por meio das quais outras pessoas interagem e publicam conteúdo, que por sua vez incorpora a característica de pessoas publicarem conteúdo, da web 2.0, sendo portanto uma evolução da mesma.

Tabela 1 - Comparação entre Web 1.0, Web 2.0 e Web 3.0.

WEB 1.0	**WEB 2.0**	**WEB 3.0**
1996 – 2004	2004 -2016	2016
Web de Hipertexto	Web Social	Web Semântica
Tim Berners Lee	Tim O'Reilly, Dale Dougherty	Tim Berners Lee
Apenas Leitura	Leitura e Escrita	Web Executável
Milhões de Usuários	Bilhões de Usuários	Trilhões de Usuários
Uni-direcional	Bi-direcional	Ambiente Virtual Multi-usuário
Empresas publicam conteúdo	Pessoas publicam conteúdo	Pessoas constroem aplicações para interação e publicação de conteúdo.
Conteúdo Estático	Conteúdo Dinâmico	Web 3.0 é curiosamente indefinida. Usa AI e aprendizado web.
Websites Pessoais	Blogs e Perfis Sociais	SemiBlog, Haystack Browser
Painéis de Mensagens	Portais de Comunidade	Fóruns Semânticos
Listas de amigos e endereços	Redes Sociais Online	Informação Social Semântica

Retirada de Choudhury et al, 2012.

2.2. Metodologia de Desenvolvimento

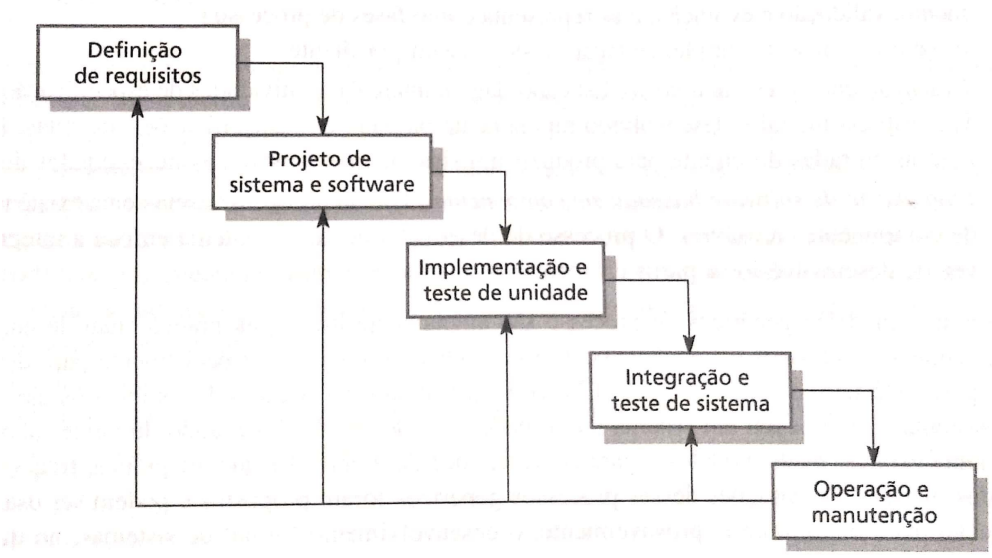
Ao longo da história dos softwares, diversas metodologias de desenvolvimento de softwares foram desenvolvidas e difundidas. Essas metodologias não são mutualmente exclusivas e, na prática, frequentemente são utilizadas em conjunto, especialmente no desenvolvimento de sistemas de grande porte. Nesse capítulo serão discutidas as principais metodologias abordadas na literatura de engenharia de software, com base nos trabalhos de Sommerville (2007), Schach (2008) e Sommerville (2011).

2.2.1. Modelo em Cascata

O modelo em cascata é o mais antigo e até hoje o mais utilizado modelo de desenvolvimento de software. Compreende uma abordagem sistemática e sequencial no desenvolvimento do software composta por cinco etapas, onde a entrada de cada etapa depende do resultado da etapa anterior, como ilustrado na Figura 1.

Comparado com outros modelos de desenvolvimento de software, ele é mais rígido e menos administrativo, apresentando assim um menor enfoque na interação entre as pessoas envolvidas no projeto. O modelo presume que o cliente participa ativamente no projeto e que sabe o que quer, e que os requisitos são bem conhecidos. Com ele, o desenvolvimento do software torna-se mais sistematizado e previsível, reduzindo assim a probabilidade de surpresas desagradáveis ao longo do projeto.

Figura 1 - Modelo em Cascata.



Retirada de Sommerville, I. Engenharia de Software. 8ª edição. Editora Pearson. Pág. 44.

Definição de requisitos: Na primeira etapa são estabelecidos os requisitos do software que se deseja desenvolver, que resulta da tradução dos objetivos do software em funções específicas, além de limitações permitidas e métricas de desempenho. Os requisitos devem ser definidos de maneira adequada para serem utilizados na próxima etapa.

Projeto de sistema e software: Na segunda etapa são projetados os seguintes atributos do sistema: estrutura de dados, arquitetura do software e interfaces. Deve-se projetar tais atributos respeitando todos os requisitos definidos e de forma a facilitar a codificação do software na próxima etapa.

Implementação e teste de unidade: Na terceira etapa o código do software é desenvolvido, resultando em instruções que podem ser executadas pelo computador. Nesse estágio, o projeto de software é realizado como um conjunto de unidades de programa, conhecidas como módulos, cada um com sua especificação. Então os módulos são testados individualmente antes de passar para a etapa de integração e teste do sistema.

Integração e teste de sistema: Na quarta etapa os módulos são integrados e o sistema construído é testado como um todo. É necessário garantir o comportamento desejado utilizando-se entradas definidas, que abranjam os requisitos especificados na primeira etapa, e comparando os resultados obtidos com os esperados. Caso os testes tenham resultados positivos, o sistema de software é liberado para o cliente.

Operação e manutenção: A etapa final consiste em instalar o sistema e o colocar em operação. A manutenção envolve melhorias na implementação, correções de erros que possivelmente passaram despercebidos nas etapas anteriores e ampliação das funções do software a medida que novos requisitos são identificados. Ela está mais ligada ao futuro do software do que ao desenvolvimento do software projetado.

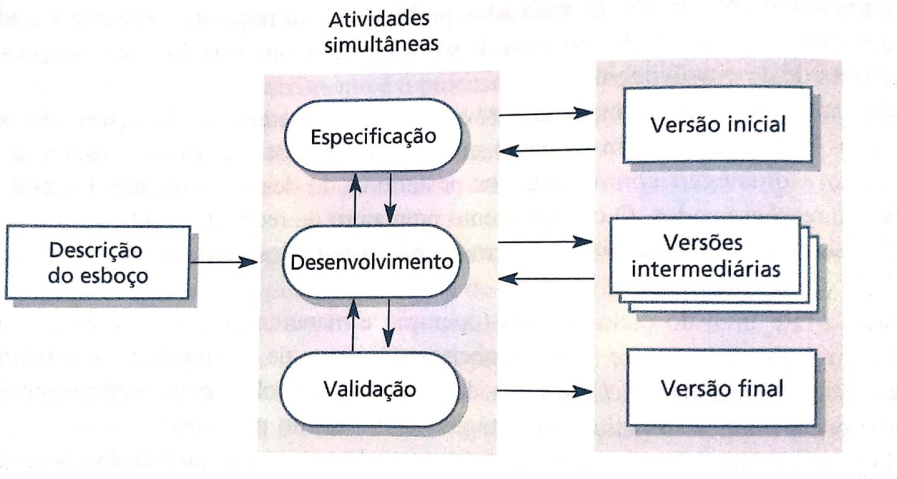
Em princípio, o resultado de cada fase consiste de um ou mais documentos aprovados, e cada fase só pode ser iniciada após a documentação e aprovação da fase anterior. Essa rigidez no processo implica em um ‘retrabalho’ significativo toda vez que os requisitos mudarem, e por isso muitas vezes partes do desenvolvimento são suspensas para que se possa prosseguir para a etapa posterior, podendo resultar em um sistema que não atenda o que o cliente deseja ou em um sistema mal estruturado, em que problemas são resolvidos posteriormente ou simplesmente ignorados.

Por isso, deve-se considerar utilizar o modelo em cascata somente apenas quando os requisitos forem muito bem compreendidos desde o início do projeto e houver poucas probabilidades de mudanças durante o desenvolvimento do sistema.

2.2.2. Modelo Evolucionário

O modelo evolucionário é baseado na idéia de desenvolvimento de uma implementação inicial, que é então exposta aos usuários, e com base em seu feedback, o resultado é refinado. Esse processo se repete diversas vezes, até que um sistema adequado seja desenvolvido. A especificação, desenvolvimento e validação são atividades intercaladas, o que permite feedbacks rápidos. Geralmente é mais eficaz que a abordagem em cascata, sendo sua maior vantagem a possibilidade da especificação ser desenvolvida de forma incremental.

Figura 2 - Modelo Evolucionário.



Retirada de Sommerville, I. Engenharia de Software. 8ª edição. Editora Pearson. Pág. 46.

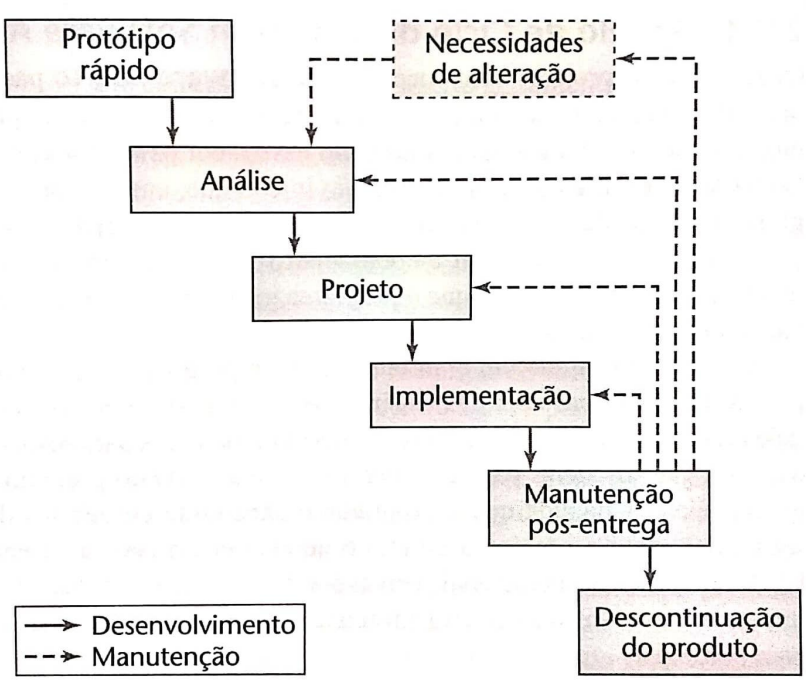
Entretanto, a abordagem evolucionária também apresenta alguns problemas. Do ponto de vista de gerenciamento, com um desenvolvimento tão rápido e iterativo não é economicamente viável produzir documentos para cada versão do sistema, e assim os gerentes não tem produtos regulares para medir o progresso. Do ponto de vista de engenharia, conforme o tamanho do software aumenta, as mudanças contínuas tendem a corromper a estrutura do software, tornando a incorporação de mudanças de software cada vez mais difícil.

Portanto, o modelo evolucionário é o melhor método de desenvolvimento para sistemas de pequeno e médio porte (até 500 mil linhas de código). A partir desse ponto, os problemas desse modelo tornam-se graves, sendo recomendada outra metodologia de desenvolvimento que suporte o estabelecimento de uma arquitetura estável do sistema de software.

2.2.3. Modelo de Prototipagem Rápida

Um protótipo rápido é um modelo operacional que é funcionalmente equivalente a um subconjunto do software final, e é a característica mais marcante do modelo de prototipagem rápida. Nesse modelo o desenvolvimento do produto é linear, partindo do protótipo rápido e passando pelas etapas de análise, projeto e implementação, após a qual o software é entregue ao cliente e é iniciada a última etapa, a manutenção pós-entrega, que irá durar até a descontinuação do software.

Figura 3 - Modelo de Prototipagem Rápida.



Retirada de Schach, S. Engenharia de Software: Os paradigmas clássico & orientado a objetos. 7ª edição. Editora McGraw-Hill. Pág. 53.

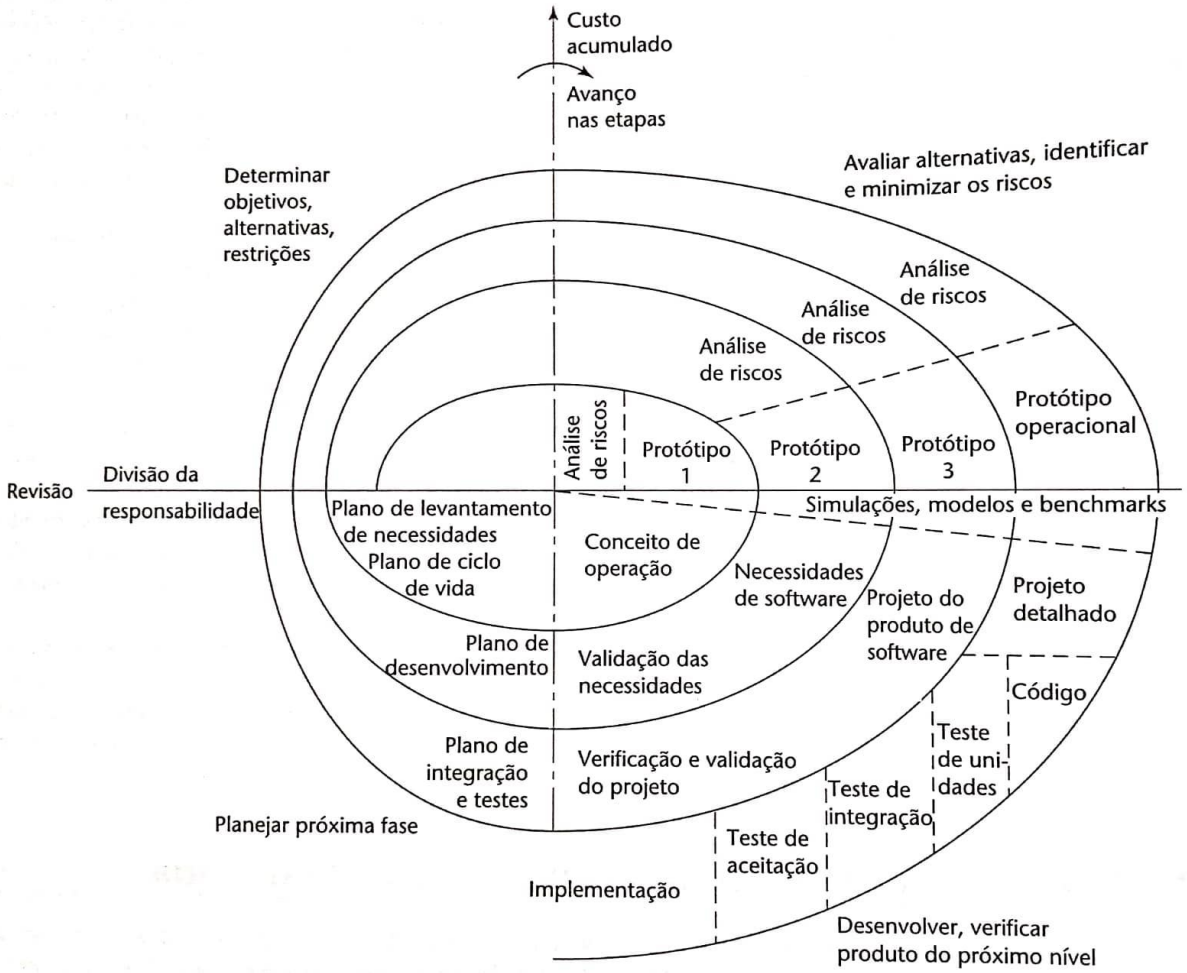
A primeira etapa no modelo de prototipagem rápida é criar um protótipo rápido e entregá-lo ao cliente e aos usuários futuros, permitindo que eles interajam e experimentem o protótipo. Quando o cliente estiver satisfeito com o protótipo rápido, ou seja, quando o protótipo cumprir com a maior parte do que é necessário, os desenvolvedores podem fazer o documento de especificações com a garantia de que o software atende às verdadeiras necessidades do cliente. Após a aprovação do protótipo rápido o processo de desenvolvimento do software é iniciado, passando pelas etapas de análise, projeto e implementação, que são semelhantes às etapas análogas no modelo cascata, e finalmente entrando na etapa de manutenção pós-entrega.

Nessa última etapa, as voltas de realimentação têm menor probabilidade de serem necessárias graças ao aprendizado obtido com o protótipo rápido, sendo esse o ponto forte desse modelo. Isso porque na etapa de análise, como o documento de especificações foi criado a partir do protótipo rápido e o protótipo foi validado por meio de interação com o cliente, espera-se que o documento de especificações resultante seja adequado. Na etapa de projeto já há uma boa percepção de como o projeto deve funcionar graças à experiência com o protótipo. E na etapa de implementação algumas falhas já foram descobertas, reduzindo assim a necessidade de corrigir o software durante ou após a implementação.

2.2.4. Modelo em Espiral

De forma semelhante ao modelo de prototipagem rápida, o modelo em espiral baseia-se na idéia de minimizar riscos pelo uso de protótipos, com a diferença que no modelo em espiral diversos protótipos são criados. De maneira simplificada ele funciona como um modelo em cascata em que cada uma das fases é precedida por uma análise de riscos onde procura-se minimizar os riscos. Caso isso não seja possível em alguma das fases, o projeto é radicalmente alterado ou encerrado imediatamente.

Figura 4 - Modelo em Espiral.



Retirada de Schach, S. Engenharia de Software: Os paradigmas clássico & orientado a objetos. 7ª edição. Editora McGraw-Hill. Pág. 63.

O modelo em espiral completo é mostrado na figura 4. A dimensão radial representa o custo acumulado até então, a dimensão angular representa o progresso ao longo da espiral e cada ciclo da espiral corresponde a uma fase.

Cada fase se inicia no quadrante superior esquerdo determinando-se os objetivos da fase, alternativas para atingir esses objetivos e restrições impostas sobre essas alternativas, resultando em uma estratégia para atingir esses objetivos.

Em seguida, no quadrante superior direito, essa estratégia é analisada sob o ponto de vista de riscos. São realizadas tentativas para minimizar todos os possíveis riscos construindo-se, em alguns casos, um protótipo. Se alguns riscos não puderem ser minimizados o projeto pode ser interrompido imediatamente, ou então, sob certas circunstâncias, pode ser tomada a decisão de se prosseguir com o projeto em uma escala muito menor.

Se todos os riscos foram minimizados com sucesso a próxima etapa de desenvolvimento é iniciada, no quadrante inferior direito. Esse quadrante do modelo espiral corresponde ao modelo cascata clássico, onde as fases avançam conforme o ciclo se distancia do centro da espiral.

Finalmente os resultados do desenvolvimento são avaliados e a fase seguinte planejada, no quadrante inferior esquerdo.

Algumas categorias de risco podem ser efetivamente testadas por meio dos protótipos. Por exemplo, restrições de tempo podem ser testadas por meio da construção de um protótipo, tomando-se medidas para verificar se o protótipo é capaz de atingir o desempenho desejado. Por outro lado, algumas categorias de riscos são mais difíceis de se testar utilizando protótipos, como por exemplo o risco de não se conseguir contratar pessoal suficiente para construir o software. Por isso é importante manter em mente que embora a prototipagem ajude a reduzir os riscos em algumas áreas, em outras ela pode não apresentar resposta alguma.

Existem também algumas restrições na aplicabilidade do modelo em espiral. Mais especificamente, o modelo destina-se exclusivamente ao desenvolvimento de software em larga escala, pois não faz sentido realizar uma análise de riscos se o custo dessa análise for comparável ao custo do projeto como um todo. Outra restrição é que no caso de ser um software contratado toda análise de riscos tem de ser realizada antes do contrato ser assinado, e não ao longo do projeto como no modelo espiral.

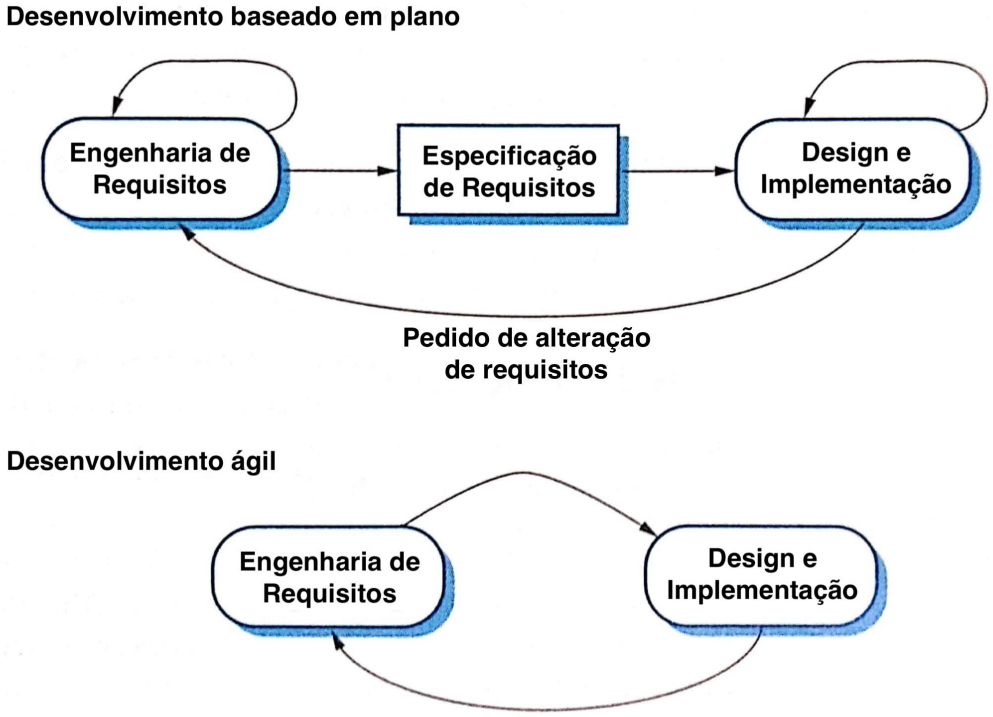
Finalmente, o ponto forte do modelo espiral, que é ser dirigido por riscos, também pode ser seu ponto fraco. Se os desenvolvedores não forem hábeis em localizar os possíveis riscos e analisá-los de forma precisa, existe o perigo de que a equipe acredite que tudo está correndo bem quando na verdade o projeto caminha para o fracasso. Por isso só é recomendado o uso desse modelo caso os membros da equipe de desenvolvimento sejam analistas de risco competentes. Mas a principal fraqueza do modelo em espiral, assim como dos modelos em cascata e prototipagem rápida, é o fato de ele assumir que o software é desenvolvido em fases discretas, quando na realidade o desenvolvimento de software é iterativo e incremental.

2.2.5. Métodos Ágeis

Hoje em dia, softwares são parte fundamental de quase todas as operações de negócios ao redor do mundo. Esses mesmos negócios operam em um ambiente global que muda rapidamente e têm que responder a novas oportunidades de mercado, variações econômicas e emergência de novos produtos e serviços competidores na mesma velocidade em que essas mudanças ocorrem. Dadas essas mudanças frequentes é praticamente impossível estabelecer um conjunto completo de requisitos estáveis do software desejado. Os requisitos iniciais inevitavelmente mudam porque é impossível prever como um sistema vai afetar as práticas atuais, como ele vai interagir com outros sistemas e quais operações deverão ser automatizadas.

Dessa forma a utilização de uma metodologia de desenvolvimento orientada por um plano fixo pode causar grandes problemas no ambiente de negócios já que, quando o software estiver disponível para uso, a razão original para o seu desenvolvimento pode ter mudado tão radicalmente que o software torna-se efetivamente inútil. Por isso, para sistemas de software de negócios, o desenvolvimento e a entrega rápida são agora os requisitos mais críticos, e muitos negócios estão dispostos a trocar a qualidade do software e o compromisso em atender a todos os requisitos pela velocidade rápida de desenvolvimento e implantação do software. Esse cenário resultou na criação de um novo tipo de metodologia de desenvolvimento: Os métodos ágeis.

Figura 5 - Comparação entre desenvolvimento baseado em plano e desenvolvimento ágil.



Retirada de Sommerville, I. Engenharia de Software. 9ª edição. Editora Pearson. Pág. 63.

Os métodos ágeis são métodos de desenvolvimento incremental em que os incrementos são pequenos e tipicamente novas versões do sistema são criadas e disponibilizadas para os consumidores a cada duas ou três semanas, a fim de conseguir rápido feedback dos usuários. Eles também buscam minimizar o tempo gasto com documentação, utilizando comunicações informais ao invés de reuniões formais com documentações escritas.

Esses métodos são projetados para produzir software útil muito rapidamente. O software não é desenvolvido como uma única unidade mas sim como uma série de incrementos, onde cada incremento inclui uma nova funcionalidade de sistema. Hoje já existem várias abordagens para o desenvolvimento rápido de software, mas todas elas compartilham algumas características fundamentais.

A primeira característica é que os processos de especificação, design e implementação são intercalados. Não há uma especificação detalhada do sistema e a documentação de design é minimizada ou gerada automaticamente pelo ambiente de programação utilizado para implementar o sistema. O documento de requisitos define apenas as características mais importantes do sistema. Além disso o sistema é desenvolvido em uma série de versões, onde o usuário final e outros interessados são envolvidos na especificação e avaliação de cada versão. Eles podem propor novos requisitos e mudanças para o software, que devem ser implementados em versões posteriores do sistema. Finalmente, as interfaces do sistema são frequentemente desenvolvidas utilizando um desenvolvimento interativo que permite que o design da interface seja rapidamente criado desenhando-se e colocando-se ícones na interface. O sistema pode então gerar uma interface baseada em web para um browser ou uma interface para uma plataforma específica, como Windows.

Atualmente existem diversos tipos de metodos ágeis, como *extreme programming*, *scrum*, *crystal*, desenvolvimento de software adaptativo e desenvolvimento orientado por recursos. Enquanto todos esses métodos ágeis são baseados na noção de desenvolvimento e entrega incrementais, eles propõem diferentes processos para alcançar isso. Entretanto, eles compartilham um conjunto de princípios baseado no Manifesto Ágil que fazem com que tenham muito em comum. Esses princípios estão descritos na tabela 2.

O sucesso dos métodos ágeis fez com que houvesse um grande interesse no uso desses métodos no desenvolvimento de softwares mais complexos e maiores. Entretanto, para alguns tipos de software como sistemas de controle de segurança críticos, onde uma análise completa do sistema é essencial, uma abordagem dirigida por um plano é a escolha certa. Também deve-se repensar o seu uso em sistemas muito grandes, pois sua efetividade depende da boa integração do time, que é mais difícil de se alcançar em times maiores. Além disso, na prática os princípios dos métodos ágeis podem ser difíceis de se cumprir em alguns casos.

Tabela 2 - Princípios dos métodos ágeis.

Princípio	Descrição
Envolvimento do cliente	Clientes devem ser profundamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar as iterações do sistema.
Entrega incremental	O software é desenvolvido em incrementos e o cliente especifica os requisitos a serem incluídos em cada incremento.
Pessoas, não processo	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Os membros da equipe devem desenvolver suas próprias maneiras de trabalhar sem processos prescritivos.
Aceite as mudanças	Tenha em mente que os requisitos do sistema vão mudar, por isso projeto o sistema para acomodar essas mudanças.
Mantenha a simplicidade	Concentre-se na simplicidade do software que está sendo desenvolvido e do processo de desenvolvimento sempre que possível trabalhar ativamente para eliminar complexidade do sistema.

Retirada de Sommerville, I. Engenharia de Software. 8ª edição. Editora Pearson. Pág. 263.

Por exemplo, o envolvimento do cliente no processo de desenvolvimento depende da existência de um cliente que pode e quer gastar tempo com o time de desenvolvimento e que pode representar todos os interessados no sistema. Além disso os membros do time podem não ter personalidades adequadas para o intenso envolvimento entre os membros que esse método requer. Outro problema pode ser a priorização de mudanças, especialmente em sistemas que possuem diversos interessados, pois geralmente cada interessado dá uma prioridade diferente a cada mudança.

Além desses problemas durante o desenvolvimento, podem surgir problemas no período de manutenção. Por um lado o cliente pode perder o interesse em manter o envolvimento no projeto após sua entrega, quebrando assim o primeiro princípio dos métodos ágeis, o de envolvimento do cliente. Por outro lado o time de desenvolvimento pode se desmanchar. Como não há documentação, esse desmanche acarreta na perda do conhecimento implícito do sistema, e sua manutenção torna-se muito mais difícil e custosa.

Portanto, é importante considerar as vantagens e desvantagens dos métodos ágeis antes de decidir utilizá-los, pois apesar de apresentarem uma ótima metodologia para sistemas de pequeno e médio porte desenvolvidos por uma equipe que trabalha bem entre si, pode apresentar problemas no desenvolvimento de softwares maiores, softwares críticos em sistemas de segurança, e também na manutenção de softwares já desenvolvidos, já que o desempenho do processo depende da integração do time de desenvolvimento entre si e com o cliente.

2.2.6. Considerações Finais

A evolução das metodologias de desenvolvimento de software aconteceu de forma empírica, surgindo de forma a corrigir os problemas que as metodologias anteriores apresentavam conforme o tipo de software desenvolvido se alterava, e por isso é importante lembrar que uma metodologia mais recente não é necessariamente a melhor em todos os casos. Para escolher qual metodologia utilizar deve-se primeiro analisar como é o software que se pretende desenvolver e qual é o contexto de sua utilização e de seu desenvolvimento.

Além disso, não é necessário que a escolha se limite a um método exclusivo. É muito comum a mistura de características de mais de um método, e a maioria dos autores defende que uma mistura entre os métodos é a melhor alternativa na maioria dos casos, focando-se no encaixe entre a metodologia escolhida e o tipo e contexto do software a ser desenvolvido.

A escolha da metodologia de desenvolvimento desse trabalho será discutida com mais detalhes no capítulo 3, onde as motivações serão explicadas e a metodologia final, que engloba características de mais de uma metodologia discutida nesse capítulo, será apresentada.

2.3. Arquiteturas

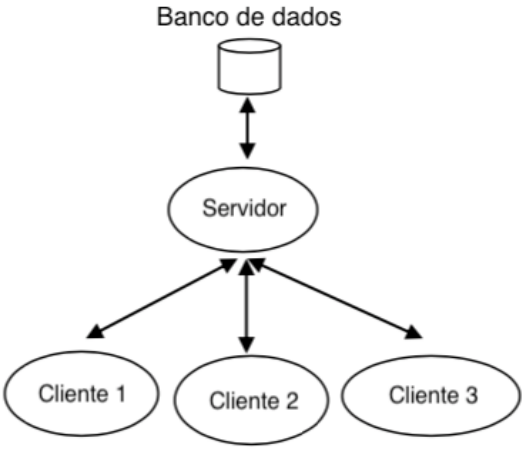
Com o avanço da tecnologia o uso da web está se tornando cada vez mais presente nas nossas vidas, principalmente por meio de aplicações. Nesse capítulo serão discutidas as principais arquiteturas utilizadas atualmente, com base nos trabalhos de Oluwatosin (2014), Oliveira (2003), e Pressman (2010).

2.3.1. Arquiteturas Cliente-Servidor

Para que a nossa interação online seja a mais agradável possível, arquiteturas de desenvolvimento web foram desenvolvidas e aprimoradas com o objetivo de garantir uma boa comunicação entre os dispositivos dos usuários (cliente) e o conteúdo disponível online, presente em bancos de dados remotos (servidor). Por isso, a arquitetura da web é conhecida como arquitetura cliente-servidor.

Essa arquitetura funciona da seguinte maneira: o cliente envia um pedido ao servidor, o servidor processa esse pedido e retorna uma resposta ao cliente. Esse processo de comunicação permite a separação de funções, o que facilita o desenvolvimento do sistema e também resulta num melhor desempenho no seu funcionamento, já que reduz a replicação de dados e facilita o compartilhamento de recursos, que podem ser distribuídos para diversos clientes a partir de apenas um servidor centralizado.

Figura 6 - Arquitetura cliente-servidor.



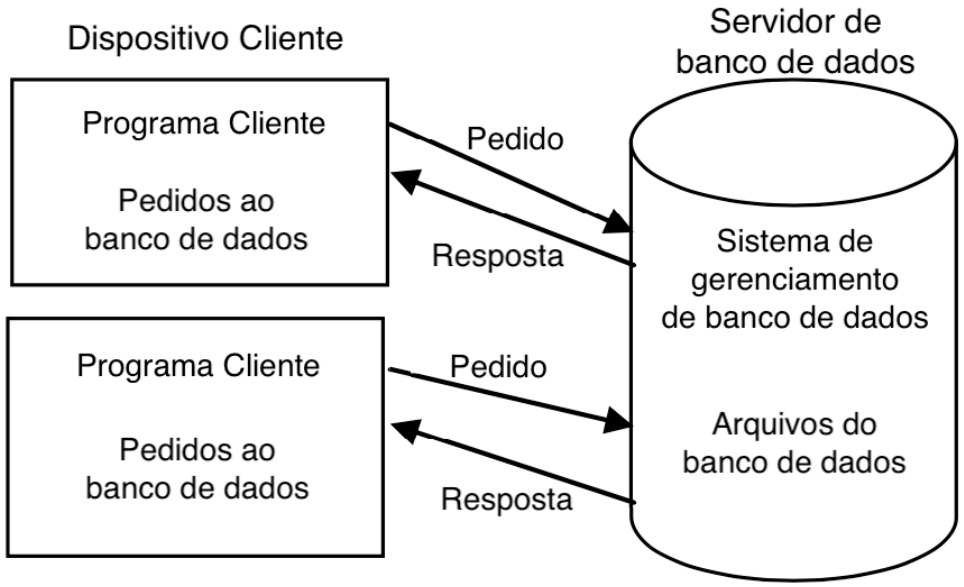
Retirada de Oluwatosin, 2014.

A arquitetura cliente servidor pode possuir diversas variações. Os dois modelos mais conhecidos atualmente são a arquitetura cliente-servidor de duas camadas e a arquitetura cliente-servidor de três camadas.

2.3.1.1. Arquitetura Cliente-Servidor de Duas Camadas

A arquitetura cliente-servidor de duas camadas é a mais simples de todas e envolve apenas o servidor de banco de dados, composto por uma potente máquina capaz de servir a vários clientes, e o dispositivo cliente, onde se encontra a interface do sistema, e ambos são conectados por uma rede (geralmente internet). O usuário executa a aplicação no seu dispositivo (cliente), que contém tanto a interface entre o usuário e a aplicação quanto a lógica de negócio, e então um pedido é enviado ao servidor de dados. O servidor então envia os dados pedidos de volta ao cliente, que os processa para então mostrar o resultado ao usuário.

Figura 7 - Arquitetura cliente-servidor de duas camadas.



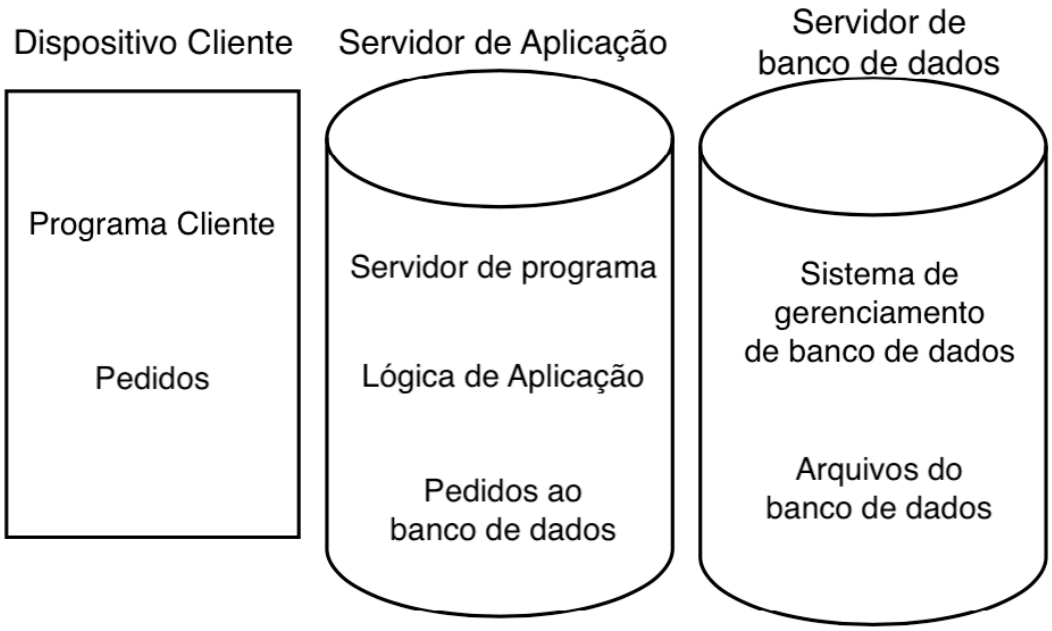
Retirada de Oluwatosin, 2014.

Essa arquitetura é uma boa solução em situações que envolvem um número de pessoas interagindo com o sistema simultaneamente na ordem de dezenas ou centenas. Quando o número de usuários excede a casa das centenas, a performance tende a deteriorar. Esta limitação é resultado do servidor mantendo a conexão “viva” com cada cliente, mesmo quando nenhum trabalho está sendo feito. Além disso, as implementações de arquiteturas de duas camadas fornecem uma limitada flexibilidade ao se alterar o funcionamento do software, pois nesse caso a nova lógica de aplicação deverá ser instalada em todos os dispositivos clientes.

2.3.1.2. Arquitetura Cliente-Servidor de Três Camadas

A arquitetura cliente-servidor de três camadas envolve o dispositivo cliente, o servidor de banco de dados e o servidor de aplicação. Nela, menos recursos e menos código são necessários no dispositivo cliente, pois ele contém apenas a lógica de apresentação. Enquanto isso, a lógica de aplicação se encontra no novo elemento da arquitetura, o servidor de aplicação, que atua como intermediário entre o dispositivo cliente e o servidor de banco de dados.

Figura 8 - Arquitetura cliente-servidor de três camadas.



Retirada de Oluwatosin, 2014.

Essa configuração permite que a camada intermediária armazene requisições de clientes em uma fila, e dessa forma o cliente pode requisitar um pedido à camada intermediária e se desconectar, pois ela vai acessar o banco de dados e retornar a resposta posteriormente. Além disso, muitas alterações no software podem ser corrigidas apenas atualizando-se a lógica de aplicação no servidor de aplicação, sem precisar de qualquer alteração nos dispositivos clientes.

Também é possível dividir a camada intermediária em mais de uma unidade com diferentes funções. Essas arquiteturas são conhecidas como n-camadas ou multicamadas, e todas funcionam utilizando o mesmo princípio de isolação de funções da arquitetura de três camadas, aumentando ainda mais o nível de isolamento e a facilidade de manutenção. Enquanto esse isolamento de funções tem como vantagem uma manutenção mais simples, eles trazem como principal desvantagem a dificuldade de implementação inicial, que aumenta conforme aumenta-se o número de camadas.

Isso porque o aumento do número de camadas é um trabalho complexo, já que a separação da interface com o usuário, a lógica de aplicação, a lógica de banco e qualquer outra divisão de dados adicional (no caso de multicamadas) muitas vezes não é trivial. Alguns processos lógicos podem aparecer em mais de uma camada, ou até mesmo todas. Ao se decidir em que camada implementar determinada função, deve-se considerar a facilidade de desenvolvimento e teste, a facilidade de administração do serviço e a escalabilidade dos servidores, levando em conta o processamento e tráfego na rede.

Por isso, é importante analisar bem os objetivos do software e capacidade de desenvolvimento antes de se escolher determinada arquitetura. A tabela abaixo fornece uma comparação entre as arquiteturas de duas camadas e três camadas com relação a algumas características chaves.

Tabela 3 - Comparação entre arquiteturas de duas e três camadas.

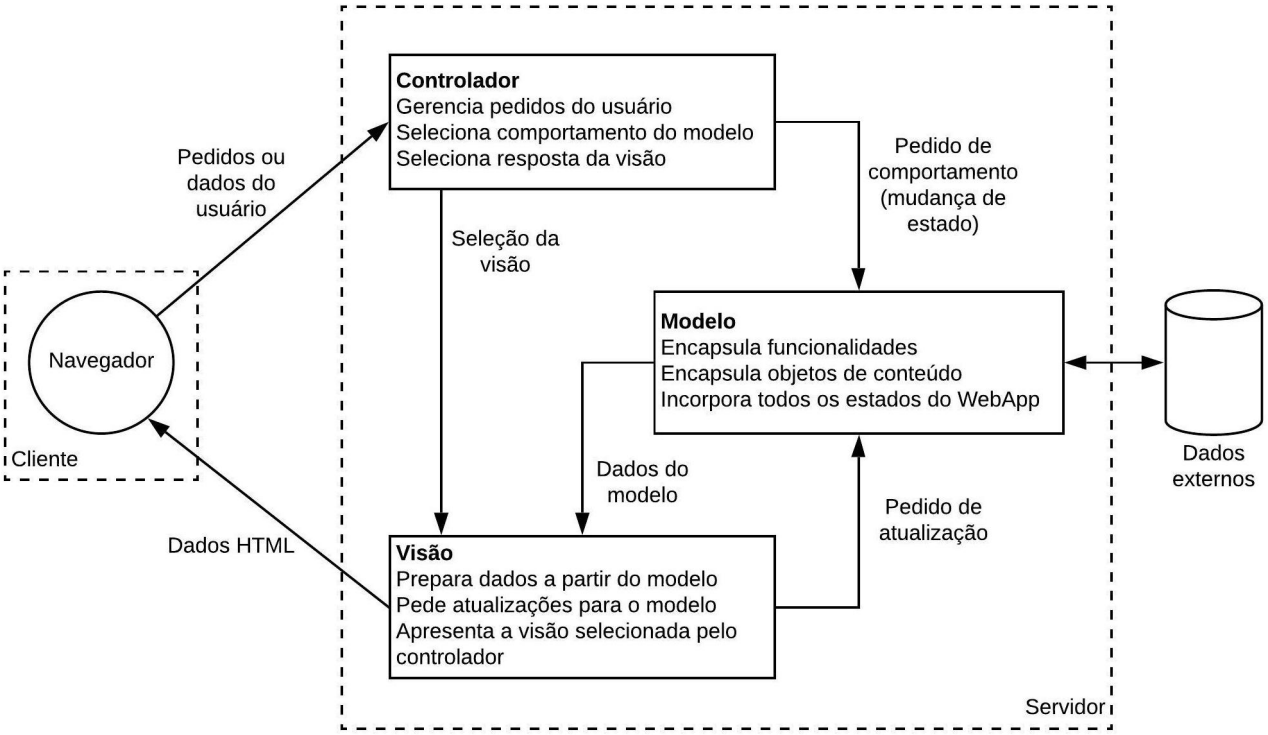
Característica	**Duas Camadas**	**Três Camadas**
Administração do sistema	Complexa (boa parte do gerenciamento no cliente)	Menos complexa (a aplicação pode ser gerenciada no servidor)
Encapsulamento de dados	Baixa (base de dados exposta)	Alta (o cliente invoca serviços ou métodos)
Performance	Média (muitas requisições SQL são enviadas através da rede)	Boa (apenas requisições de serviços são feitas entre cliente e servidor)
Escalabilidade	Baixa (gerenciamento limitado de links de comunicação com clientes)	Excelente (pode-se distribuir a carga de processamento entre diferentes servidores)
Reusabilidade	Baixa	Alta
Facilidade de desenvolvimento	Alta	Baixa (apesar de várias ferramentas de desenvolvimento disponíveis no mercado já oferecerem facilidades)

Retirada de Oliveira, H. Pág. 29.

2.3.2. Arquitetura Modelo-Visão-Controlador (MVC)

A arquitetura MVC é um tipo de arquitetura para aplicações web que simplifica a implementação e o reuso de código por meio do desacoplamento entre interface, navegação e comportamento da aplicação. O modelo contém todo conteúdo específico da aplicação e lógica de processamento como objetos de conteúdo, acesso a dados externos e funções específicas do software. A visão contém todas as funções específicas da interface e permite a apresentação de conteúdo e lógica de processamento, mostrando ao usuário dados sobre os objetos, acessos a dados externos e funções utilizadas pelo usuário final. O controlador gerencia o acesso ao modelo e a visão a partir dos pedidos e dados enviados pelo usuário e coordena o fluxo de dados entre eles, atualizando a visão com dados do modelo baseado na entrada do usuário.

Figura 9 - A arquitetura MVC.



Retirada de Pressman, R. 7ª Edição. Pág. 387. (2010)

Como mostra a figura, os pedidos ou dados do usuário são manipulados pelo controlador, que então seleciona o objeto de visão adequado ao pedido do usuário e envia um pedido de alteração de estado ao modelo, que por sua vez executa essa alteração e atualiza a visão com a alteração em efeito.

2.3.3. Considerações Finais

As duas principais arquiteturas de aplicações web são a arquitetura cliente-servidor e a arquitetura MVC. É importante notar que elas não se excluem, e normalmente são utilizadas de forma conjunta, como é o caso desse trabalho, apresentando divisões diferentes que ajudam a separar o desenvolvimento em partes menores e simplificando assim o processo de desenvolvimento e manutenção do software.

A aplicação dessas duas arquiteturas ao desenvolvimento desse trabalho será discutida com mais detalhes no capítulo 3, onde o diagrama do funcionamento do software será apresentado e será dividido em relação a cada uma das duas arquiteturas utilizadas.

3. Desenvolvimento

Como discutido no capítulo 2, é preciso analisar o tipo e o contexto do software a ser desenvolvido antes de se escolher a metodologia a ser utilizada. No caso desse trabalho, o objetivo é desenvolver uma forma automática de comparar todos os políticos em relação às ideologias e crenças políticas de cada usuário. Apesar de não ser um desafio trivial, também não há, a primeira vista, nenhum grande obstáculo para seu desenvolvimento, dada a existência e disponibilidade de ferramentas para auxiliar no desenvolvimento da aplicação.

Dessa forma, percebeu-se que seria possível fornecer no mínimo a função principal da aplicação antes da próxima eleição, em 2018, e então tornou-se essencial a conclusão de uma versão utilizável pelo usuário antes dessa eleição. Por isso uma restrição na data de lançamento da aplicação foi criada, e tal data foi definida para o dia 7 de Setembro de 2018, um mês antes da eleição. Assim, o objetivo inicial no desenvolvimento foi o de implementar e testar a função principal da aplicação (comparação de políticos), e então implementar outras funções secundárias, de forma priorizada, almejando-se construir a aplicação mais completa possível dentro do tempo disponível.

Sabendo que o software a ser desenvolvido não é de uma complexidade muito grande e que o conjunto de requisitos finais pode variar dependendo do fluxo do desenvolvimento, a escolha de uma abordagem evolucionária tornou-se bastante atraente. Entretanto, alguns aspectos da abordagem em cascata também pareceram coerentes ao projeto, por dois motivos principais: porque os requisitos eram bem conhecidos e porque um planejamento de arquitetura em que as funções se encaixariam ao longo do desenvolvimento garantiria a facilidade de manutenção da aplicação no longo prazo, permitindo e facilitando assim a evolução da aplicação após a eleição.

Por isso, uma abordagem mista foi adotada, com características tanto do modelo em cascata quanto do modelo evolucionário. Nela, inicialmente formula-se os apenas os requisitos principais, de forma priorizada, para que possam ser seguidos ou alterados ao longo do desenvolvimento. Então, planeja-se a arquitetura a ser seguida no desenvolvimento, incluindo nesse planejamento a definição da arquitetura e também a forma como será implementada. A partir desse planejamento inicia-se o desenvolvimento do software, e desse ponto em diante desenvolve-se o software de maneira evolucionária, onde a especificação, o desenvolvimento e a validação ocorrem de forma alternada conforme o desenvolvimento evolui, sem uma ordem específica a ser seguida. Essa abordagem pode ser visualizada na figura a seguir.

Figura 10 - Metodologia de desenvolvimento.

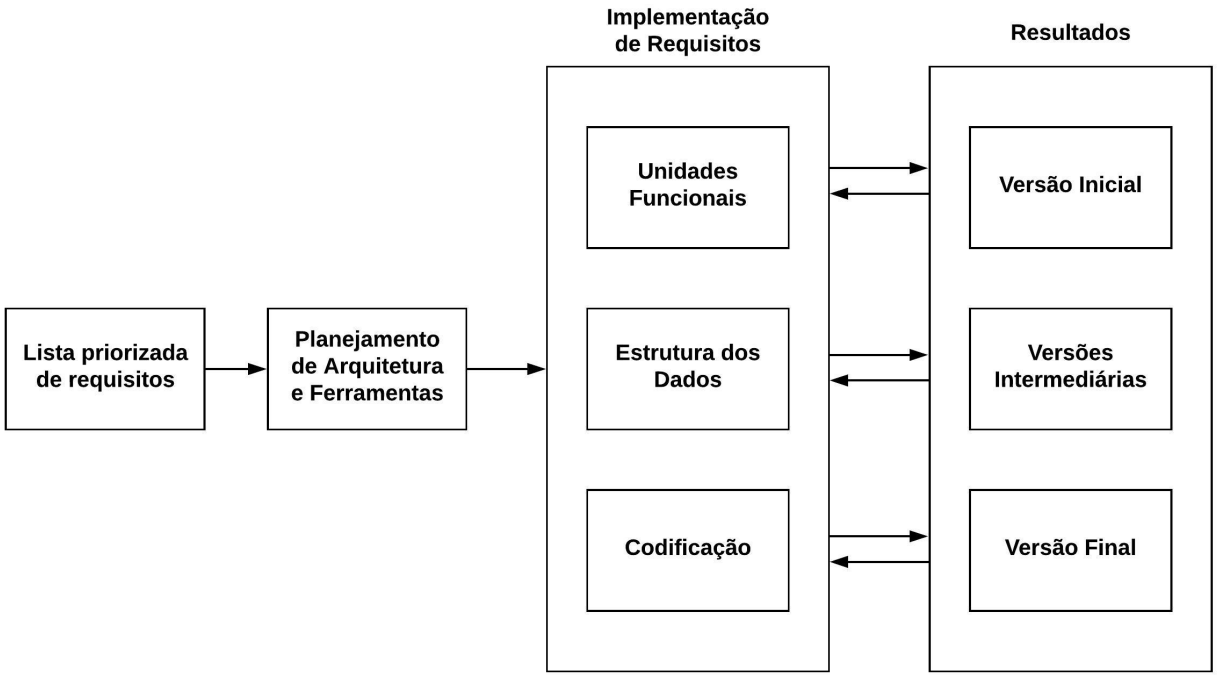


Figura produzida pelo autor.

3.1. Definição de Requisitos

O primeiro passo para o desenvolvimento é a definição de uma lista priorizada dos principais requisitos. Essa lista deve ser bem simples e sucinta, já que os requisitos podem mudar no futuro, e serve apenas para ser um guia do desenvolvimento. A lista para esse trabalho foi:

1. Comparação de políticos
2. Salvar dados do usuário
3. Comunicação entre usuários na forma de comentários nos projetos

Tal lista contém as principais funções que são desejadas à aplicação em sua forma ideal. Ela é simples o suficiente para guiar o desenvolvimento do trabalho sem consumir muito tempo em seu desenvolvimento, e por isso foi utilizada. Além disso, um requisito em paralelo à essa lista é o de construir uma aplicação com o visual mais atraente possível, e a melhora visual foi uma atividade que aconteceu logo após a finalização de cada requisito.

3.2. Planejamento de Arquitetura e Ferramentas

Após a definição dos requisitos principais, foi então planejada a arquitetura do sistema a ser desenvolvido. Essa planejamento foi dividido em duas fases. Primeiramente, escolheu-se as arquiteturas a serem tomadas como base, e então definiu-se as ferramentas a serem utilizadas para cumprir as exigências das arquiteturas base. As arquiteturas base escolhidas foram a arquitetura cliente-servidor de três camadas e a arquitetura modelo-visão-controlador.

A arquitetura cliente-servidor de três camadas foca na separação física (por dispositivos) da presença de código, e foi escolhida porque permite que a aplicação resultante no dispositivo cliente seja muito mais leve, já que concentra boa parte das funcionalidades no servidor de aplicações, e muitas vezes isso também permite que a aplicação seja atualizada sem qualquer mudança no código presente no dispositivo cliente. Por fim, essa separação acarreta num maior nível de organização do software, permitindo uma evolução muito mais escalável do mesmo.

A arquitetura modelo-visão-controlador foca na separação de funções do código em três tipos de função: interface (visão), navegação (controlador) e comportamento da aplicação (modelo). Ela foi escolhida porque facilita muito a evolução da aplicação desenvolvida, já que tal separação de funções aumenta a reusabilidade das mesmas, além de também elevar o nível de organização do código em desenvolvimento.

Uma vez definidas as arquiteturas a se tomar como base, chega a hora de definir quais ferramentas escolher para cumprir as exigências das arquiteturas. É importante que as ferramentas escolhidas atendam às duas arquiteturas escolhidas simultaneamente. Felizmente, como as duas arquiteturas escolhidas estão entre as mais utilizadas atualmente, encontrar ferramentas que atendam a esses requisitos não apresenta um grande desafio. As principais ferramentas escolhidas foram Framework7, PHP e ArangoDB, e as respectivas descrições que seguem foram baseadas nas informações encontradas nas páginas oficiais de cada ferramenta.

Framework7 é um framework de software aberto e gratuito projetado para se desenvolver aplicações web ou aplicativos móveis híbridos (que podem ser transformados em aplicativos Android ou iOS por meio da utilização de um encapsulamento, como Cordova), além de também poder ser utilizado como uma ferramenta de prototipação. O grande diferencial do Framework7 é que esse desenvolvimento pode ser realizado utilizando-se apenas HTML (*Hypertext Markup Language*), CSS (*Cascading Style Sheets*) e JavaScript, que são as linguagens mais simples e comuns do desenvolvimento web, permitindo assim o desenvolvimento de uma aplicação avançada e completa sem a necessidade de se aprender linguagens específicas e complexas para tal.

PHP (*Hypertext Preprocessor*) é uma linguagem de código aberto de uso geral, mas é especialmente usada em desenvolvimento web, sendo uma das mais utilizadas para esse fim desde

sua criação até os dias atuais. O seu maior diferencial é a simplicidade, principalmente para iniciantes, pois sua estrutura permite mesclar código PHP com código HTML, facilitando muito o desenvolvimento, que adquire um fluxo mais natural ao alternar entre código de cliente (HTML) e código de servidor (PHP). Apesar da simplicidade, o PHP também oferece uma enorme gama de recursos mais avançados que foram desenvolvidos com o passar dos anos, além de contar com uma ampla comunidade que possibilita o acesso a uma enorme rede de conhecimento, disponível em fóruns oficiais e não oficiais.

ArangoDB é um banco de dados código aberto multimodelo, o que significa que pode trabalhar com dados tanto no formato de documentos quanto no formato de grafos ou no formato chave-valor. Ele utiliza uma linguagem muito semelhante ao SQL (*Structured Query Language*), chamada de AQL (*Arango Query Language*), que também permite acessar ou modificar dados de forma simples e direta. Entretanto, por não ser SQL, o ArangoDB permite uma flexibilidade muito maior na estrutura dos dados utilizados, abrindo por exemplo a possibilidade de dois documentos do mesmo tipo possuirem atributos diferentes, o que resulta em um nível de escalabilidade muito maior.

Dentro do paradigma Cliente-Servidor o Framework7 fica responsável pelo processamento no cliente, juntamente com o visual da interface construído em HTML, CSS e Javascript. As funções em PHP ficam responsáveis pelo funcionamento do servidor de aplicação, abrangendo tanto a API (*Application Programming Interface*) de comunicação com o Framework7 quanto as funções da aplicação propriamente dita. Já o ArangoDB fica responsável pelo acesso e gerenciamento do servidor de banco de dados. Essa distribuição, juntamente com a arquitetura cliente-servidor final da aplicação, pode ser visualizada na figura a seguir.

Figura 11 - Divisão da aplicação na arquitetura Cliente-Servidor.

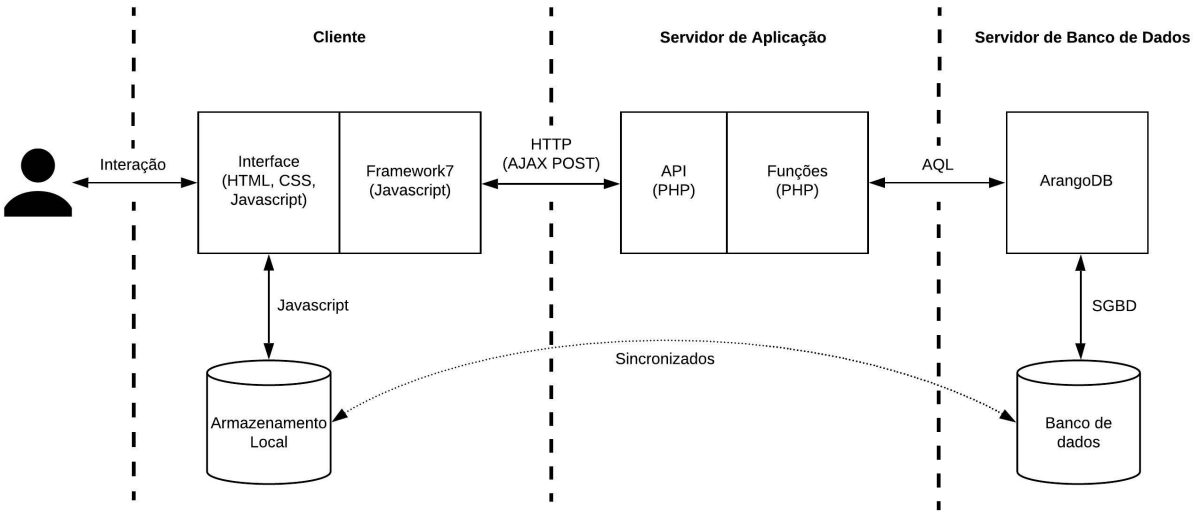


Figura produzida pelo autor.

Dentro do paradigma MVC a visão corresponde apenas ao visual das páginas da interface, construídas com HTML, CSS e Javascript. O Framework7 fica responsável pelo gerenciamento dos pedidos do usuário e pela seleção da resposta da visão, enquanto também conversa com a API para selecionar o comportamento do modelo através da chamada de funções PHP, realizando assim o papel de controlador de forma conjunta. As funções em PHP ficam responsáveis pelo encapsulamento das funções e o ArangoDB fica responsável pelo encapsulamento de objetos de conteúdo e pelo armazenamento dos estados da aplicação, realizando assim o papel do modelo de forma conjunta. Essa configuração, juntamente com a arquitetura MVC final da aplicação, pode ser visualizada na figura a seguir.

Figura 12 - Divisão da aplicação na arquitetura MVC.

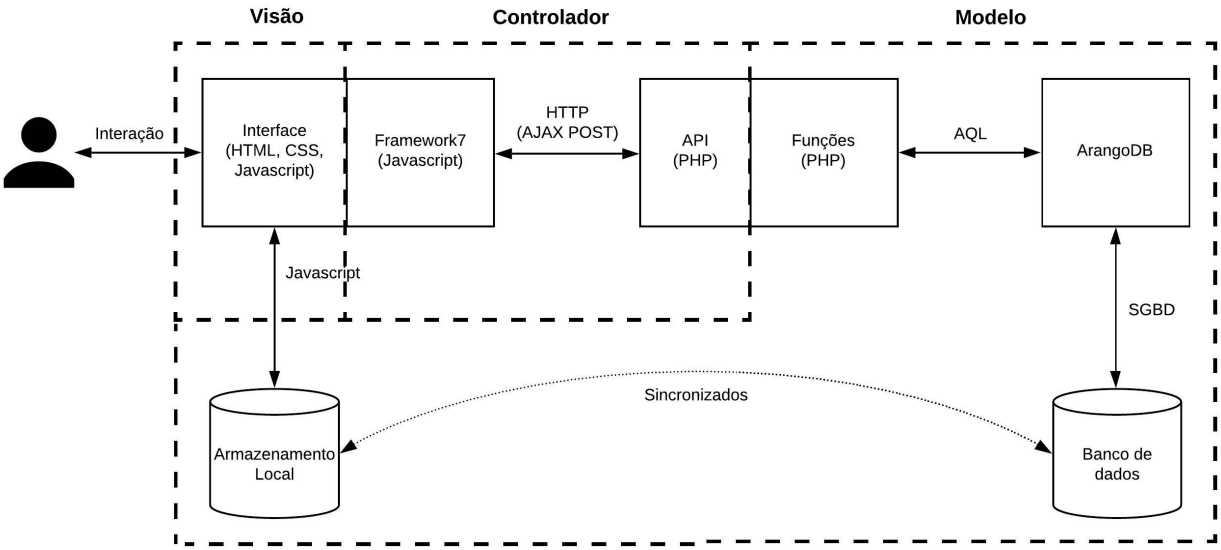


Figura produzida pelo autor.

3.3. Implementação

Uma vez definidas a arquitetura e as ferramentas a serem utilizadas, a implementação da aplicação teve início. O fluxo de atividades a ser desenvolvido, considerando todas as três funções principais descritas na lista inicial, pode ser observado na figura a seguir, onde as atividades estão separadas de acordo com a função que elas cumprem, refletindo o ponto de vista do usuário.

Figura 13 - Fluxo de atividades geral.

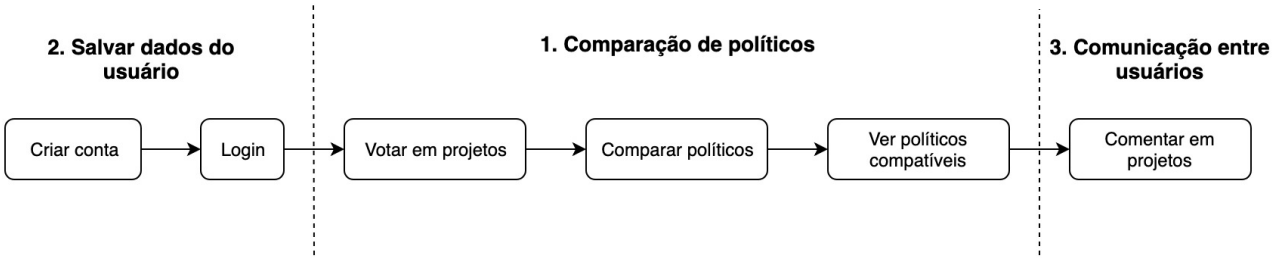


Figura produzida pelo autor.

Por ser a função principal, a comparação de políticos foi implementada primeiro. Em seguida, foi implementada a função de “salvar dados do usuário”, e por último foi implementada a função de comunicação entre usuários. Durante a implementação de cada função principal as atividades foram desenvolvidas na sequencia indicada.

Como as ferramentas utilizadas foram projetadas para seguir o paradigma MVC e a implementação do projeto utilizando tais ferramentas só é possível respeitando-se esse paradigma, a preocupação em seguir esse conceito não foi necessária. Por isso, e para evitar repetições, os fluxos de dados serão descritos apenas em relação ao paradigma cliente-servidor.

3.3.1. Comparação de Políticos

3.3.1.1. Votar em Projetos

Como a prioridade inicial do desenvolvimento foi implementar a função de comparação de políticos (como definido nos requisitos iniciais), a primeira sub-função a ser implementada foi a de votar em projetos. Para isso, implementou-se o fluxo de atividades que pode ser observado abaixo.

Figura 14 - Fluxo de atividades para votar em projetos.

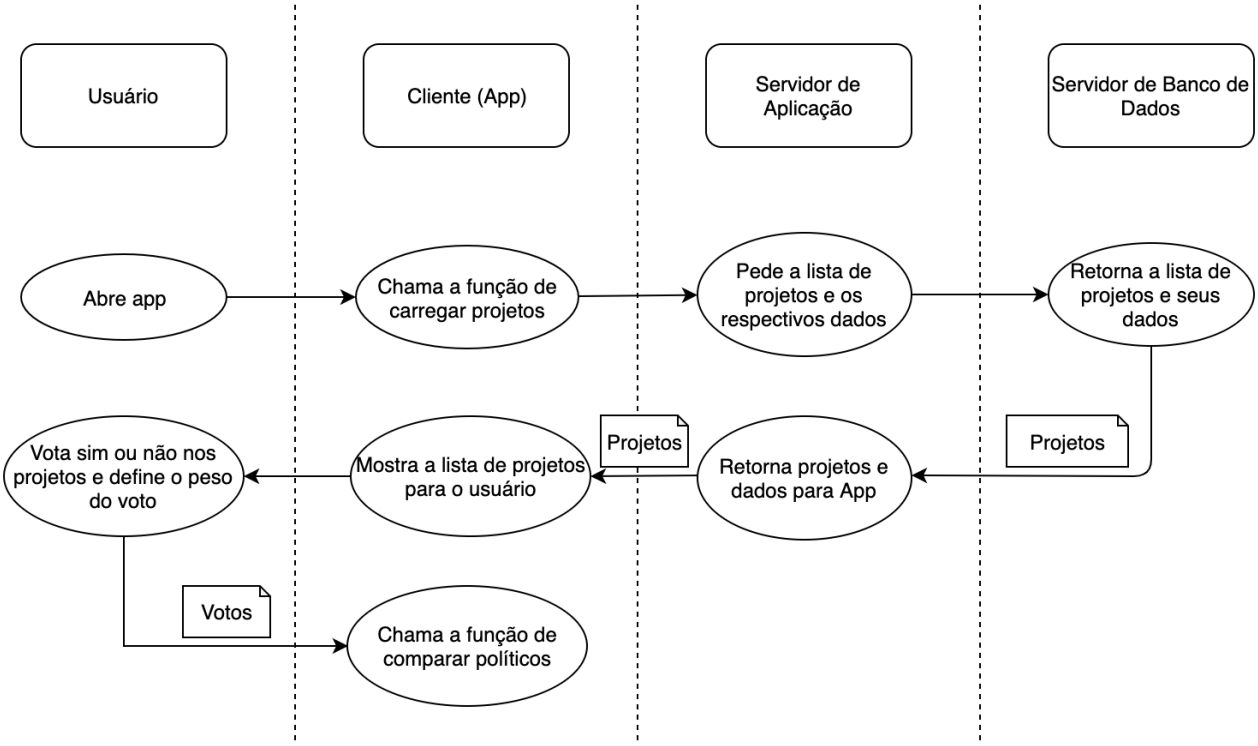


Figura produzida pelo autor.

O primeiro passo é exibir os projetos visando fácil visualização e entendimento para o usuário, carregando os dados do modelo referentes aos projetos a serem exibidos. Juntamente com os projetos, são exibidos também os botões que são utilizados pelo usuário para efetuar o voto e os campos numéricos utilizados pelo usuário para definição do peso do voto.

Para isso, é necessário que cada projeto tenha todo o conjunto de dados a ser utilizado armazenado previamente no modelo. Esses dados abrangem desde o nome oficial do projeto até os políticos envolvidos nesse projeto, para que possam ser acessados posteriormente e seus votos comparados com os votos do usuário. Além disso, diversos dados como título, resumo e descrição do projeto, obtidos por meio de pesquisa do autor sobre os mesmos, são utilizados para facilitar o entendimento do usuário sobre a que se refere o projeto, já que apenas os dados oficiais são geralmente insuficientes para que o público em geral possa compreender o significado do projeto.

A estrutura dos dados de um projeto, armazenados no modelo, assim como a tela resultante utilizada pelos usuários para votar nos projetos, podem ser observadas nas figuras a seguir.

Figura 15 - Estrutura dos dados de um projeto.

Projeto
Título Resumo Descricao Imagem NomeAutor ImagemAutor NomeCamara LinkCamara NomeSenado LinkSenado Envolvidos[] {KeyPolitico, Nome, Voto}

Figura produzida pelo autor.

Figura 16 - Tela de votos em projetos.

Figura produzida pelo autor.

3.3.1.2. Comparar Políticos

Então passou-se à implementação da principal função da aplicação, a comparação de políticos. Para isso, implementou-se o seguinte fluxo de atividades.

Figura 17 - Fluxo de atividades para comparar políticos.

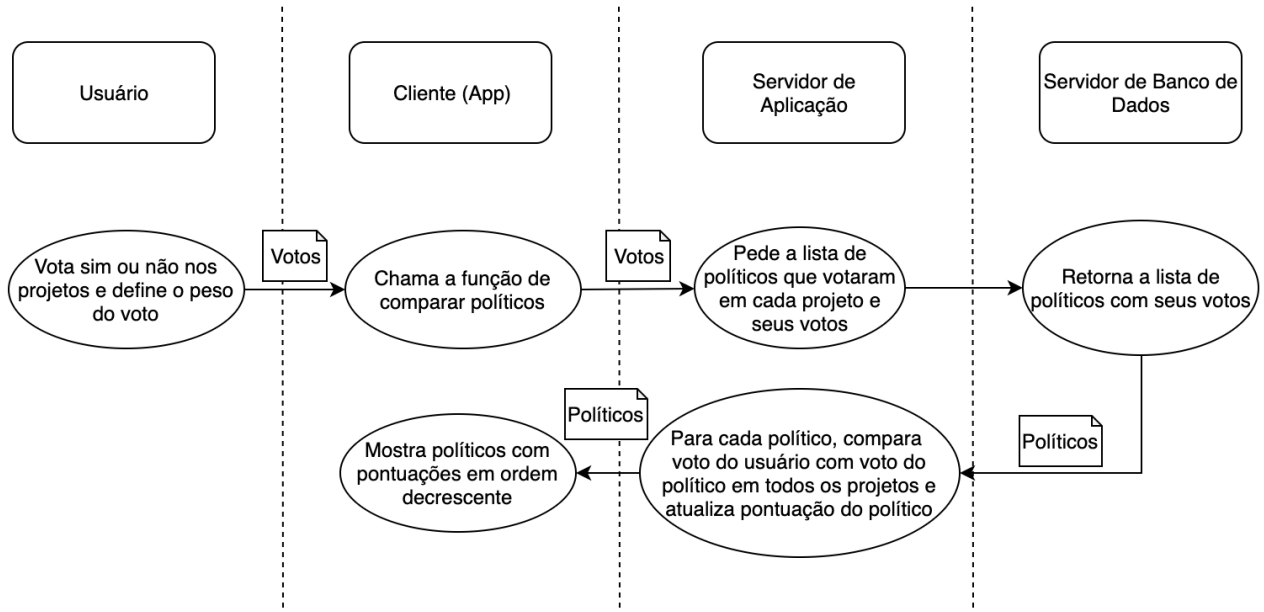


Figura produzida pelo autor.

A comparação de políticos envolve bastante lógica e processamento, principalmente no servidor de aplicação, que é onde as comparações ocorrem de fato e são calculadas as pontuações dos políticos. A figura abaixo detalha melhor esse estágio da comparação de políticos.

Figura 18 - Comparação de políticos no servidor de aplicação.

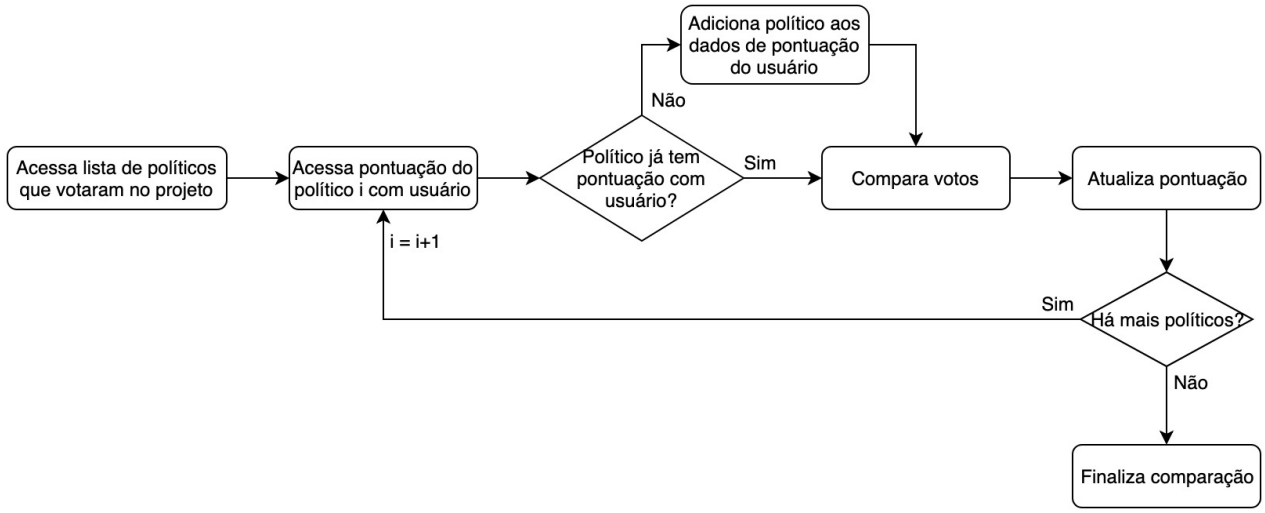


Figura produzida pelo autor.

O primeiro passo é acessar a lista de políticos que votaram no projeto, ou seja, o vetor "Envolvidos[]" da estrutura de dados do projeto. Então acessa-se a pontuação do primeiro político da lista e verifica-se se o político já possui uma pontuação com o usuário (devido a projetos comparados anteriormente). Se ele ainda não possui pontuação com o usuário, adiciona-se o político a lista de políticos pontuados com o usuário, e então compara-se o voto do político com o do usuário e atualiza-se a pontuação do político. Caso o político já tenha pontuação com o usuário, basta comparar os votos e atualizar sua pontuação. Esse procedimento é realizado para cada político da lista de envolvidos com o projeto em comparação, que só é finalizada quando todos os políticos envolvidos tiverem seus votos comparados com o voto do usuário e sua pontuação finalizada.

Para isso, é necessário que cada político tenha todo o conjunto de dados a ser utilizado armazenado previamente no modelo. Esses dados abrangem o nome do político, seu partido, sua UF (Unidade Federativa), seu cargo, e sua foto. Além disso, também foi associado a cada político todos os projetos em que o mesmo votou, para que o usuário possa entrar no perfil de um político em específico e verificar quais foram os votos do político em cada um dos projetos em que participou.

Após atribuir as pontuações de cada político, resta apenas ordenar os políticos de acordo com as pontuações dos políticos, em ordem decrescente. Finalmente, exibe-se os políticos que mais combinam com o usuário, da maior nota para a menor. O principal aspecto a ser considerado nessa fase é a forma de apresentação dos políticos, que dados e que design utilizar para que a visualização do resultado seja simples, direta e agradável ao usuário. Para isso, conceitos de IHC (Interação Humano-Computador) foram utilizados, tanto nessa etapa quanto no design de todas as telas da aplicação.

A estrutura dos dados de um projeto, armazenados no modelo, e a tela onde os usuários visualizam os políticos resultantes podem ser observadas nas figuras a seguir.

Figura 19 - Estrutura dos dados de um político.

Politico
Nome Partido UF Cargo Foto ProjetosVotados[] {KeyProjeto, Voto}

Figura produzida pelo autor.

Figura 20 - Tela de visualização de políticos.

Figura produzida pelo autor.

3.3.2. Armazenamento de Dados dos Usuários

3.3.2.1. Criar Conta

Como definido na lista priorizada de requisitos, a segunda função a ser implementada é a de armazenamento de dados do usuário. Para tal, são necessárias duas funções intimamente relacionadas: a função de criar uma conta e a função de fazer login. Primeiramente foi implementada a função de criar conta, implementando-se um fluxo de atividades que pode ser observado na figura a seguir.

Figura 21 - Fluxo de atividades para criar conta.

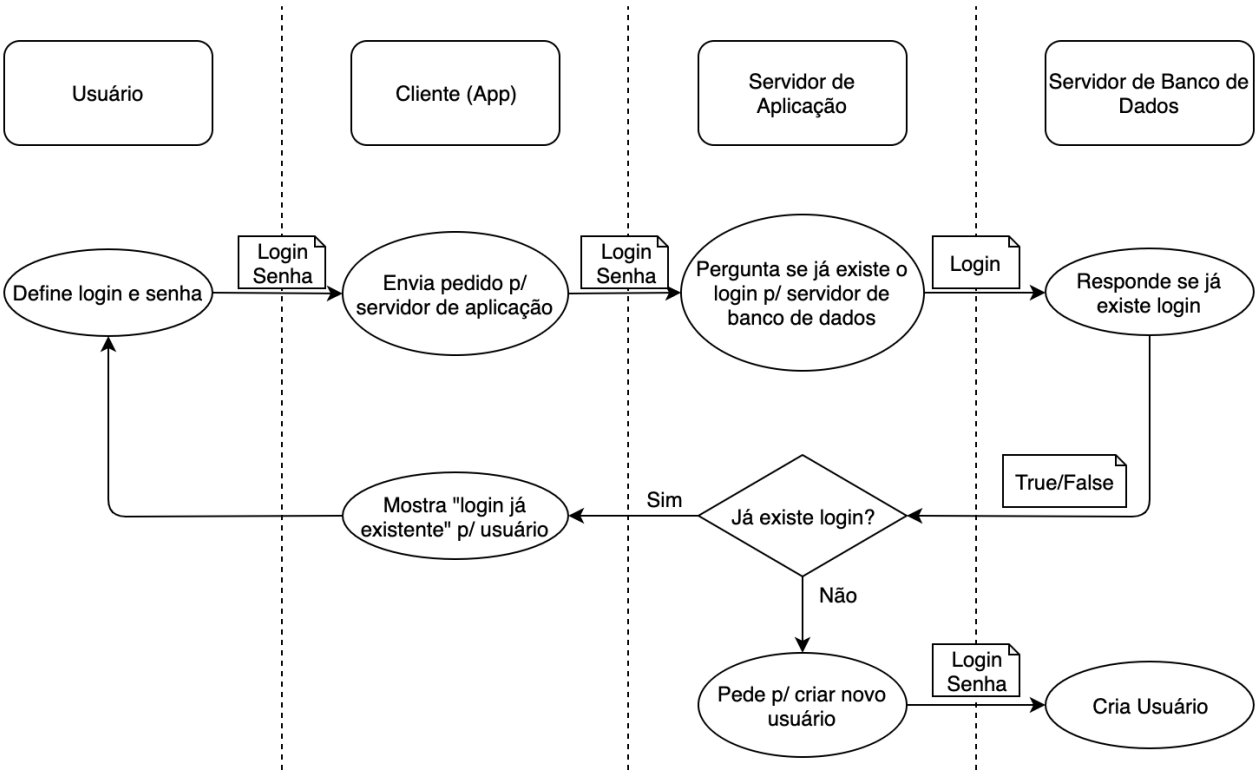


Figura produzida pelo autor.

Assim que o usuário seleciona a opção de criar conta, uma tela é exibida com os campos de definição de usuário (ou *username*) e senha. Quando os campos são preenchidos e enviados pelo usuário, é realizada uma consulta ao banco de dados e, caso o *username* já exista o usuário é alertado e deve então escolher um *username* diferente. Caso o username ainda não exista, o usuário é então cadastrado no banco de dados.

Para que o cadastro seja realizado, é necessário armazenar um conjunto de dados no modelo referentes ao usuário em questão. Esse conjunto deve possuir obrigatoriamente os dados de *username* e senha do usuário, para que seus dados possam ser reconhecidos e acessados, e também os dados que serão utilizados durante sua experiência na utilização da aplicação. Esses dados abrangem desde nome e sobrenome, que são utilizados para mostrar ao usuário que o mesmo está logado em sua conta, até os projetos em que o usuário votou, os seus votos nesses projetos, os pesos que o usuário atribuiu a esses projetos e as pontuações dos políticos envolvidos nos projetos em que o usuário votou.

Além disso, também adicionou-se os dados de votos de usuários aos projetos armazenados, para possibilitar a exibição da quantidade de votos que determinado projeto recebeu, além de quais votos são esses, revelando assim a aceitação ou rejeição de um projeto para todos os usuários em função de seus votos individuais. A estrutura dos dados de um usuário, armazenados no modelo, pode ser observada na figura a seguir, assim como as alterações realizadas nos dados dos projetos, onde os atributos em cinza representam os dados que já estavam presentes e os atributos em preto representam os dados adicionados (votos ou reações dos usuários).

Figura 22 - Estrutura dos dados de um usuário e alterações na estrutura dos dados de um projeto.

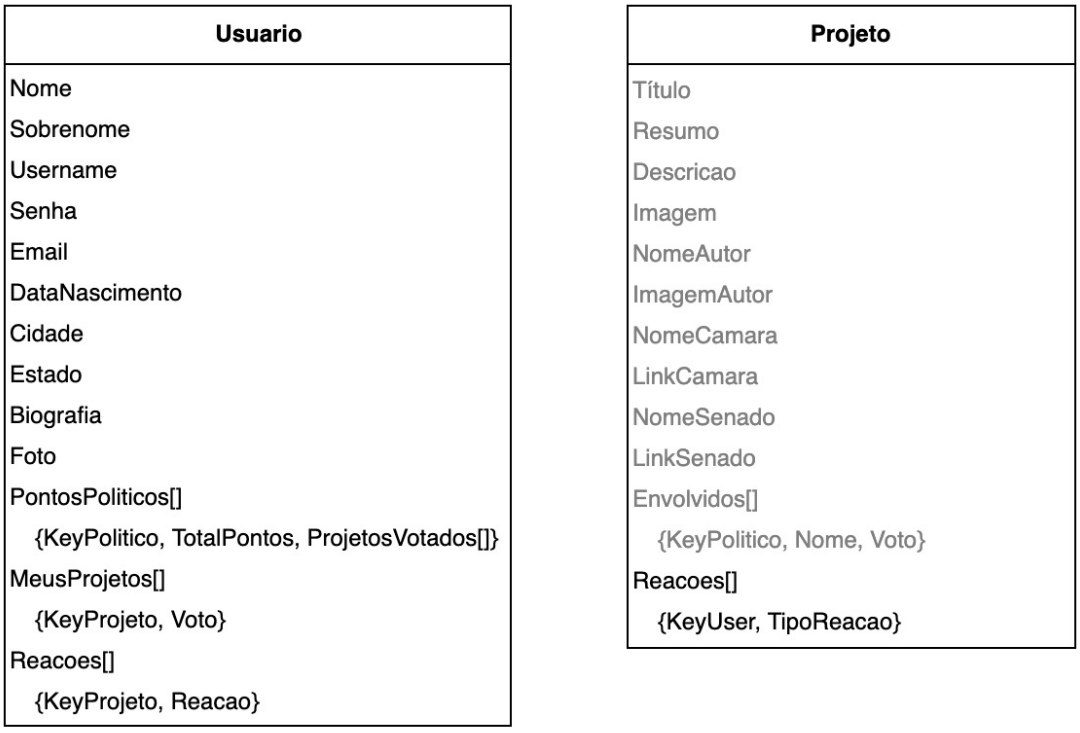


Figura produzida pelo autor.

Um detalhe sobre essa figura é que o vetor “ProjetosVotados[]” (dentro do vetor “PontosPoliticos[]”) não foi definido na mesma, pois sua estrutura é a mesma utilizada e já definida na classe “Politico”, sendo assim um vetor de pares {KeyProjeto,Voto}.

3.3.2.2. Fazer Login

Então passou-se à implementação da função de login. Para isso, implementou-se um fluxo de atividades que pode ser observado na figura a seguir.

Figura 23 - Fluxo de atividades fazer login.

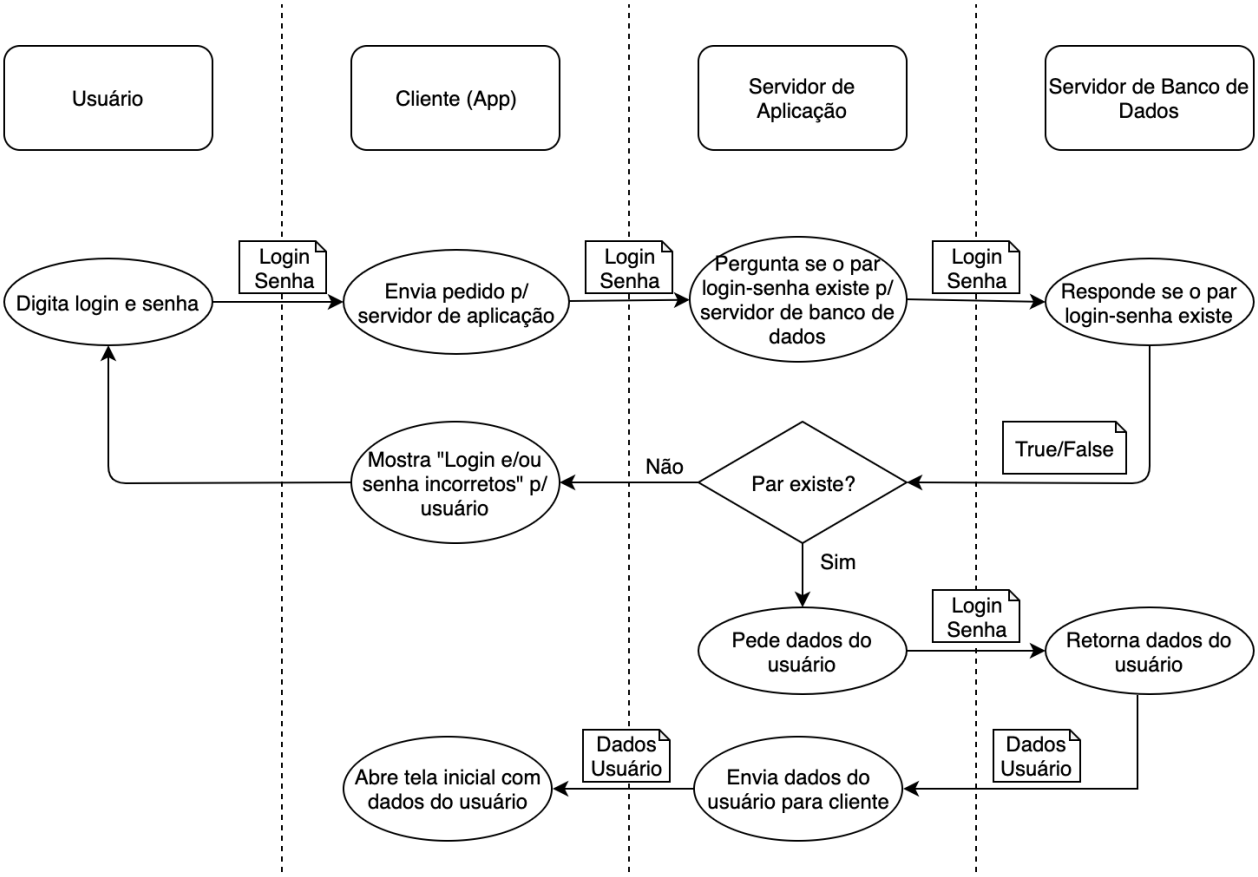


Figura produzida pelo autor.

Assim que o usuário seleciona a opção de fazer login, uma tela é exibida com os campos de entrada de usuário (ou *username*) e senha. Quando os campos são preenchidos e enviados pelo usuário, é realizada uma consulta ao banco de dados e, caso o *username* e a senha enviados correspondam a um usuário cadastrado no banco de dados, o processo é completado e os dados do usuário em questão são carregados. Caso contrário, o usuário é alertado e deve entrar com o *username* e senha corretos.

Como os dados a serem utilizados na etapa de login já foram definidos na etapa de criação de conta, não foi necessário alterar a estrutura dos dados. Assim, conclui-se a função de armazenamento dos dados dos usuários, e inicia-se o desenvolvimento da terceira função, a de permitir interações entre os usuários.

3.3.3. Interações entre Usuários

Finalmente a implementação do último requisito definido na lista priorizada de requisitos inicial, o de interação entre usuários, foi iniciada. Para isso um sistema de comentários, respostas, curtidas e notificações (similar aos utilizados nas redes sociais atualmente) foi utilizado. O fluxo de atividades referente a utilização desse sistema pode ser observado na figura a seguir.

Figura 24 - Fluxo de atividades das interações entre usuários.

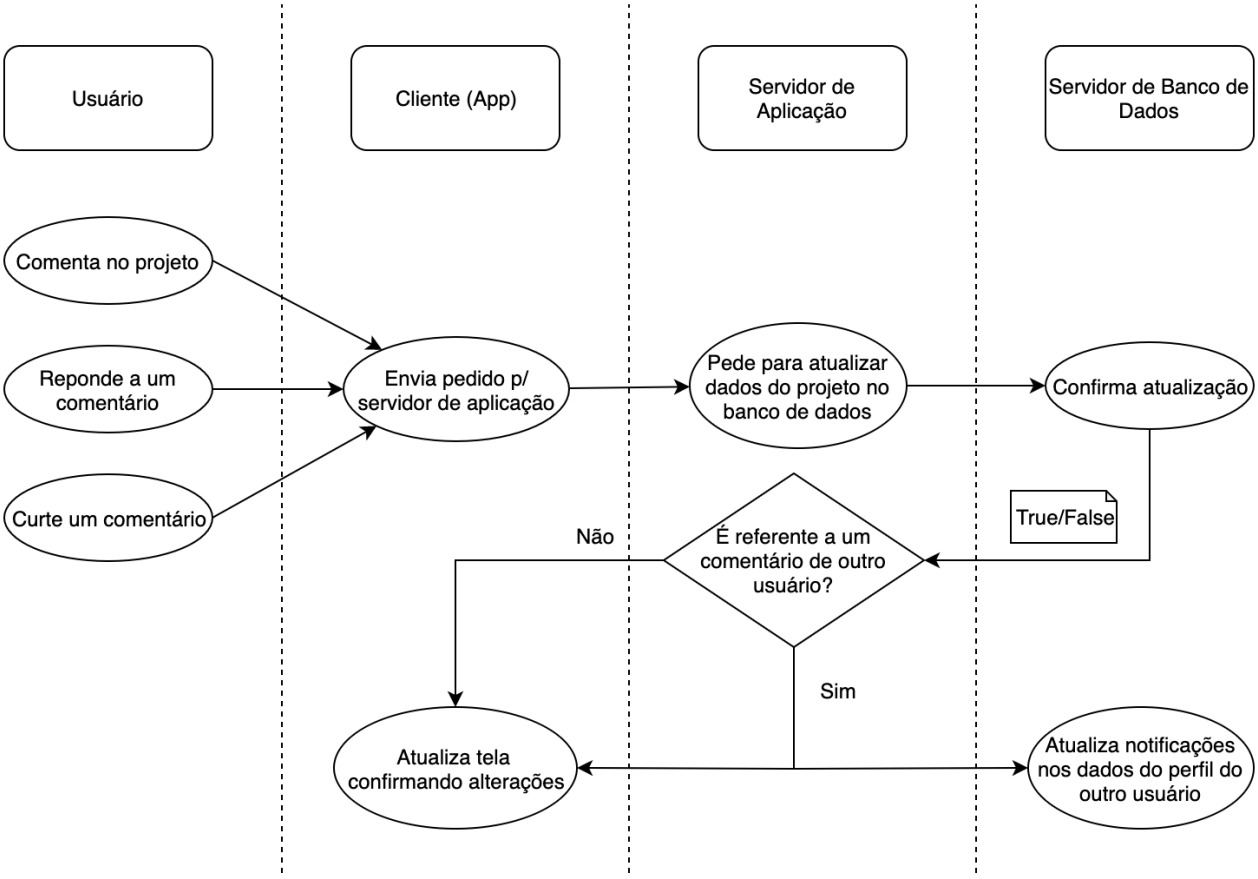


Figura produzida pelo autor.

Para utilizar essas funções, o usuário deve primeiramente abrir a página de detalhes do projeto, clicando tanto no nome, na foto ou no título do projeto. Além disso, um botão com um desenho de balão de conversa foi adicionado com a mesma função, indicando a possibilidade de comunicação com outros usuários ao utilizá-lo.

Na tela de detalhes do projeto, logo após as informações referentes ao projeto, são exibidos os comentários já realizados por usuários, e um campo para que o usuário utilizando a aplicação

possa inserir o seu comentário fica visível na parte inferior da tela da aplicação. Além disso, em baixo de cada comentário de outros usuários ficam disponíveis as opções de curtir o comentário e de responder o comentário. Sempre que um usuário responde ou curte o comentário de outro usuário, este é notificado sobre o ocorrido, e quando clica na notificação é levado diretamente ao comentário em questão, para que possa ver sobre qual comentário se trata e ler a resposta se for o caso, podendo também responder de volta. Assim, permite-se uma comunicação de duração indefinida, onde qualquer um pode participar, promovendo o debate sobre os assuntos que mais interessam ao país.

As alterações realizadas nos dados dos projetos e dos usuários podem ser observadas na figura a seguir, onde as linhas em cinza representam os dados que já estavam presentes e as linhas em preto representam os dados adicionados para permitir a interação entre os usuários.

Figura 25 - Alterações nas estruturas dos dados de projetos e usuários.

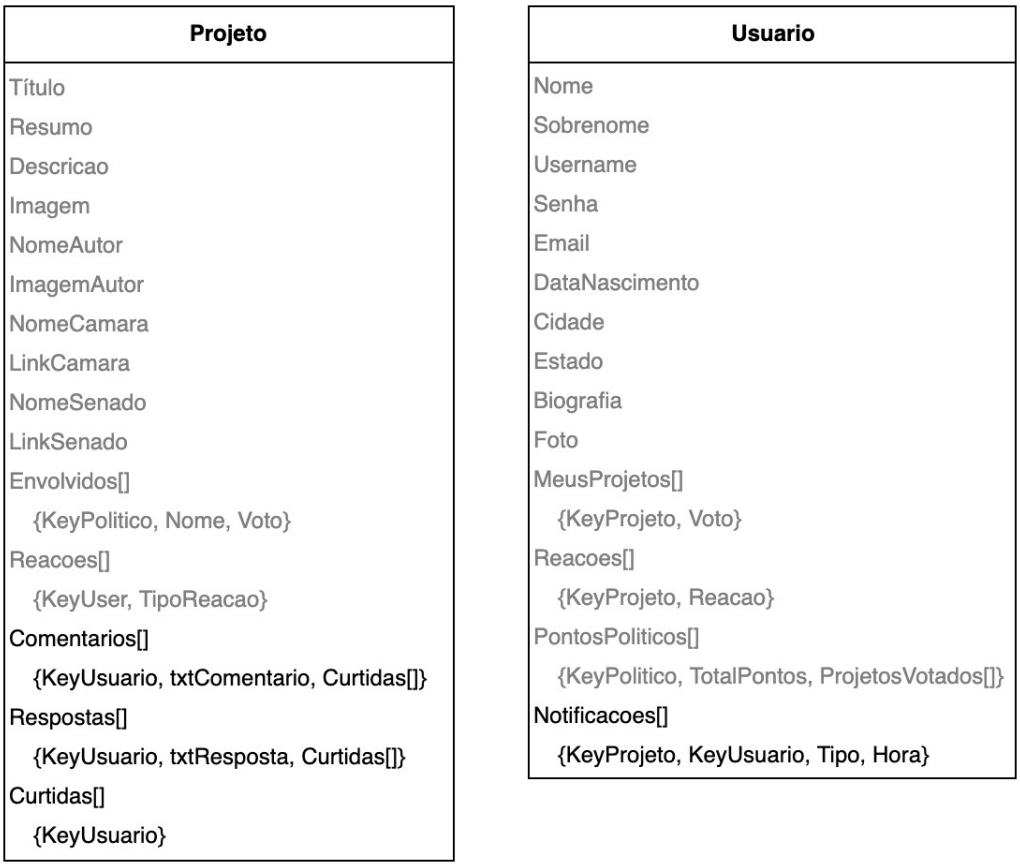


Figura produzida pelo autor.

Um detalhe importante nessa figura é que o vetor “Curtidas[]” aparece no mesmo nível dos outros atributos apenas para fins de visualização, mas ele pode existir somente dentro dos vetores “Comentarios[]” ou “Respostas[]”, já que são os únicos elementos que podem ser curtidos, enquanto que os projetos em si não podem ser curtidos, apenas comentados e votados positivamente ou negativamente.

3.4. Estrutura Final dos Dados

Ao final da implementação das três funções principais definidas na lista priorizada de requisitos inicial, os dados utilizados pela aplicação ficaram divididos em três classes principais: Projeto, Usuário e Politico. Os atributos contidos em cada uma dessas classes, em sua versão final, podem ser observados na figura abaixo.

Figura 26 - Estrutura final dos dados.

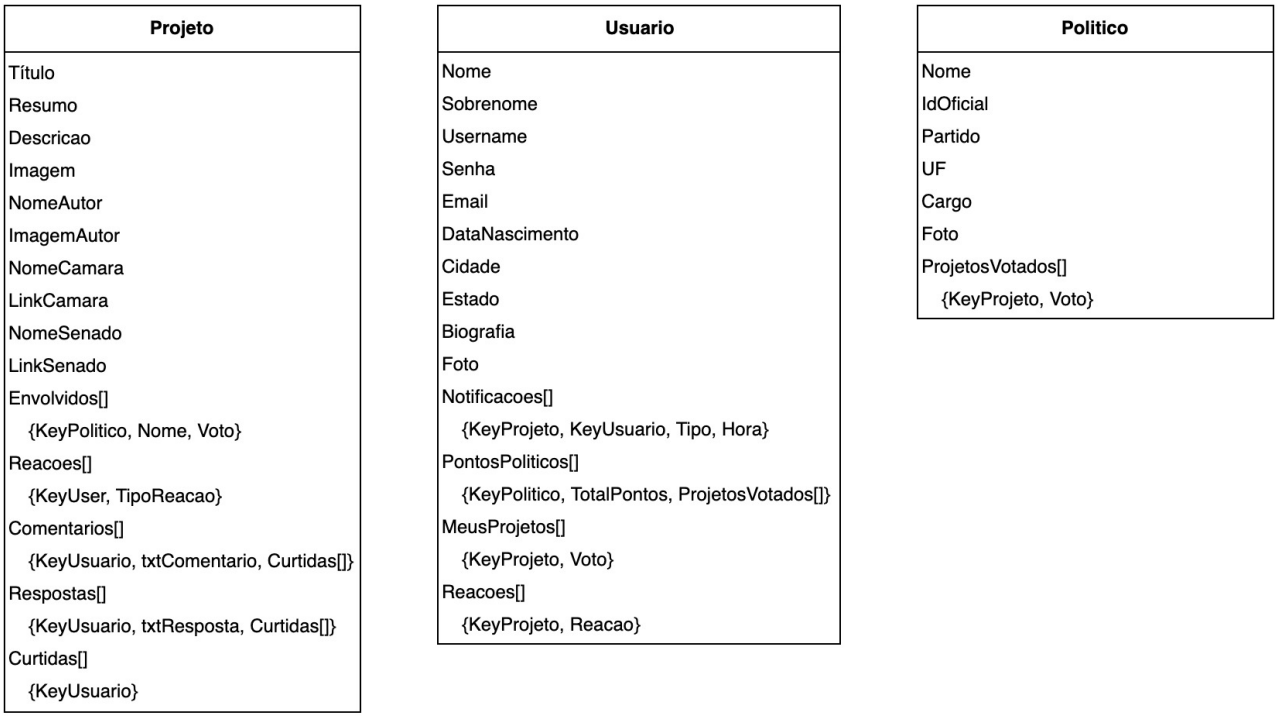


Figura produzida pelo autor.

Nota-se que, apesar do número de atributos ser grande, a maior parte deles são dados básicos utilizados apenas para informar o usuário sobre os projetos e políticos, que muitas vezes o mesmo está descobrindo pela primeira vez ao utilizar a aplicação, e também dados sobre o próprio usuário, que permitem a criação de perfil pessoal que outros usuários possam ver.

Já os atributos utilizados nas três funções principais, apesar de representarem uma quantidade menor, são bem mais complexos, sendo geralmente vetores contendo múltiplos atributos, podendo inclusive incluir outros vetores, como é o caso dos vetores “Comentarios[]”, “Respostas[]”, que contém vetores “Curtidas[]” dentro de si, e do vetor “PontosPoliticos[]”, que contém o vetor “ProjetosVotados[]” dentro de si.

Utilizou-se então o software Apache Cordova para encapsular a aplicação no formato Android, disponível no endereço https://play.google.com/store/apps/details?id=com.politik.

4. Testes e Resultados

Após o desenvolvimento da aplicação, chega a hora de testar a mesma para verificar se os objetivos do trabalho foram alcançados. Como discutido no primeiro capítulo, o objetivo desse trabalho é auxiliar as pessoas a comparar todos os candidatos em que podem votar, tomando como base as suas ações dentro do congresso nos mandatos atuais e anteriores, de forma que essa comparação seja mais fácil, rápida, precisa e abrangente do que os métodos atuais.

Por isso a melhor forma de se testar os resultados é perguntar aos usuários qual sua percepção sobre a facilidade, rapidez, precisão e abrangência da comparação de políticos realizada pela aplicação. Para a realização dessa pesquisa foi utilizado um formulário com cinco perguntas, que foi enviado a todos os usuários que cadastraram um e-mail ao criar a sua conta. Para que a pesquisa fosse o mais simples possível para o usuário, cada pergunta foi acompanhada de cinco alternativas de múltipla escolha, que podem ser observadas nas figuras com os resultados dos testes. As perguntas utilizadas foram as seguintes:

1. Qual o nível de dificuldade encontrado por você ao utilizar a aplicação?
2. Para você, qual a abrangência da comparação realizada pela aplicação?
3. Para você, qual a precisão da comparação realizada pela aplicação?
4. Quanto tempo você levou para realizar a comparação entre todos os políticos com a aplicação?
5. Quanto tempo você levaria para realizar a mesma comparação entre todos os políticos sem a aplicação?

Os resultados da primeira pergunta, que trata da facilidade de uso da aplicação, podem ser observados na figura a seguir.

Figura 27 - Resultado da pergunta sobre a facilidade de utilização da aplicação.

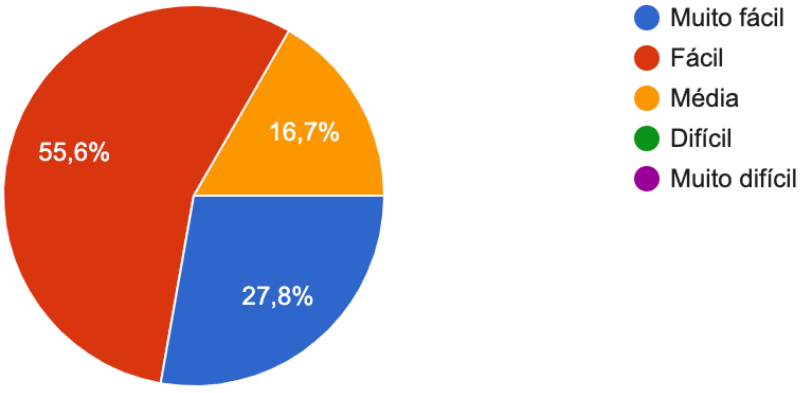


Figura produzida pelo autor.

Após a análise desse resultado, fica evidente que o objetivo de realizar uma comparação entre os políticos de forma simples e fácil foi atingido, já que a grande maioria dos usuários responderam que a aplicação de de utilização fácil ou muito fácil.

Os resultados da segunda pergunta, que trata da abrangência da comparação da aplicação, podem ser observados na figura a seguir.

Figura 28 - Resultado da pergunta sobre a abrangência da comparação da aplicação.

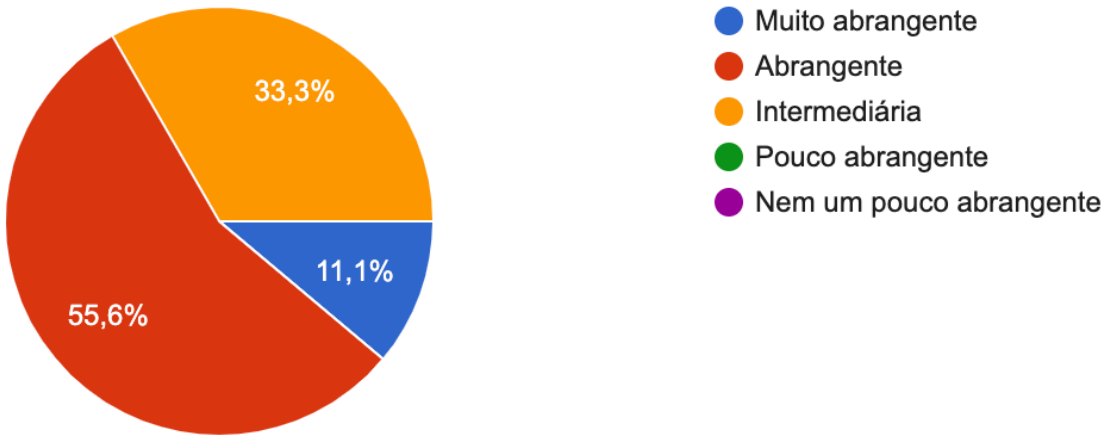


Figura produzida pelo autor.

Após a análise desse resultado, percebe-se que a percepção de abrangência da comparação varia entre “intermediária” e “muito abrangente”, com a classificação “abrangente” correspondendo à maior parte das respostas. Assim, pode-se dizer que o objetivo de realizar uma comparação abrangente entre os políticos foi atingido, apesar de ainda haver espaço para melhorias nesse sentido.

Os resultados da terceira pergunta, que trata da precisão da comparação da aplicação, podem ser observados na figura a seguir.

Figura 29 - Resultado da pergunta sobre a precisão da comparação da aplicação.

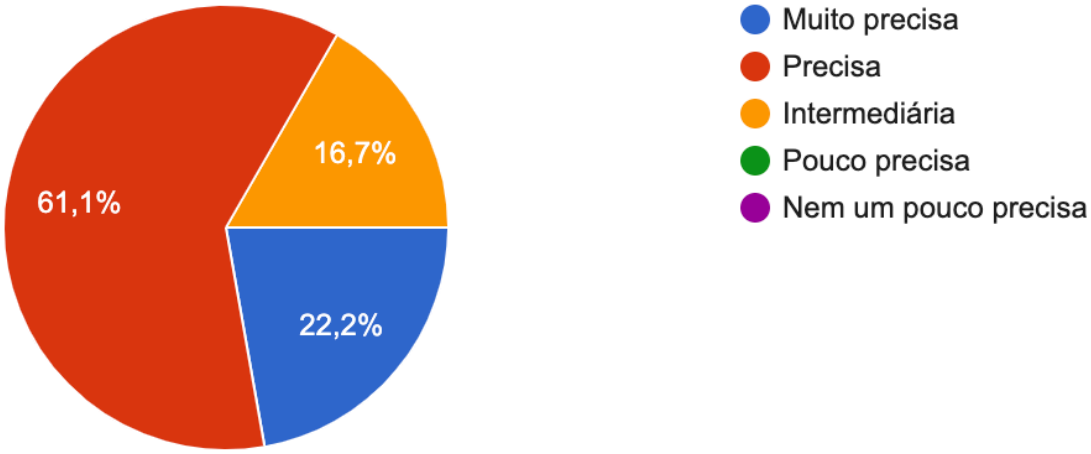


Figura produzida pelo autor.

Após a análise desse resultado, percebe-se que a percepção de precisão da comparação varia entre “intermediária” e “muito precisa”, com a classificação “precisa” correspondendo à maior parte das respostas. Assim, pode-se dizer que o objetivo de realizar uma comparação precisa entre os políticos foi atingido, apesar de ainda haver espaço para melhorias nesse sentido.

Os resultados das duas últimas perguntas, que tratam do tempo utilizado para a comparação entre os políticos utilizando-se a aplicação e sem utilizar a aplicação, podem ser observados nas figuras a seguir.

Figura 30 - Resultado da pergunta sobre o tempo de comparação dos políticos utilizando a aplicação.

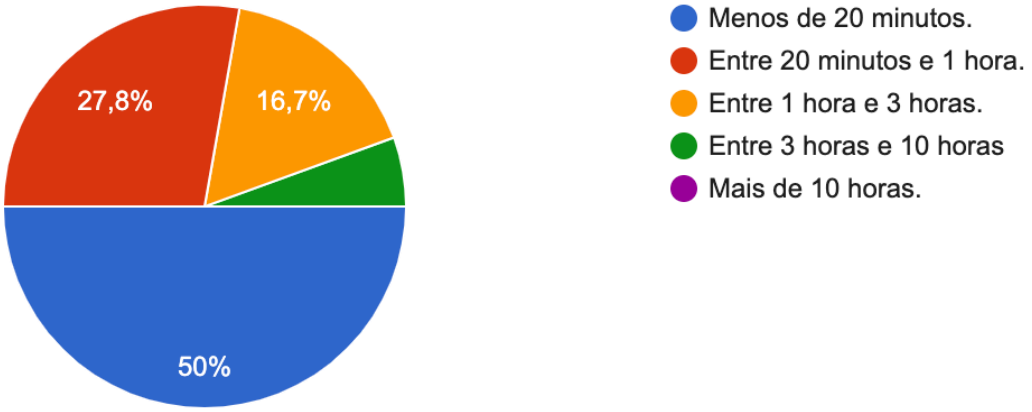
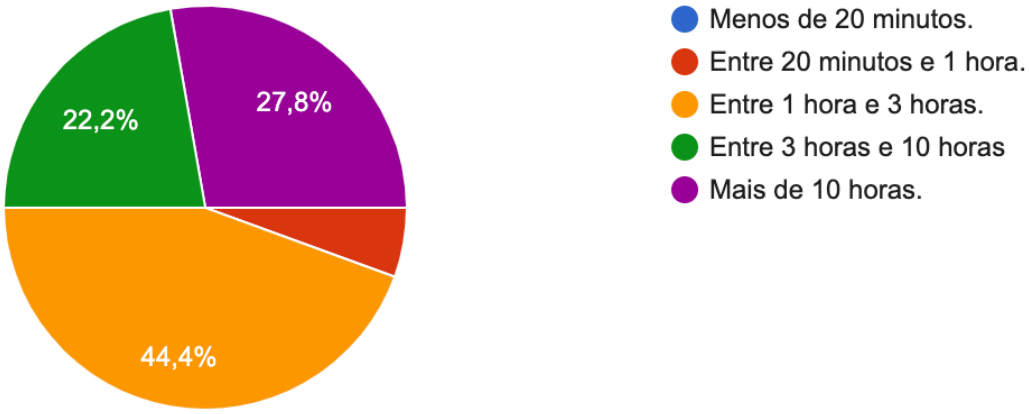


Figura produzida pelo autor.

Figura 31 - Resultado da pergunta sobre o tempo de comparação dos políticos sem utilizar a aplicação.
Figura produzida pelo autor.



Após a análise desses resultados, fica evidente que o objetivo de realizar uma comparação rápida entre os políticos foi atingido, já que metade dos usuários responderam que a comparação utilizando a aplicação durou menos de 20 minutos, e mais que 90% responderam um valor menor que 3 horas. Enquanto isso, que o intervalo de 1 a 3 horas foi a resposta mais frequente sobre o tempo mínimo estimado pelos usuários para a comparação sem utilizar a aplicação, e metade dos usuários responderam mais que 3 horas, ultrapassando a marca das 10 horas em 27,8% das vezes.

5. Conclusão

A proposta do trabalho foi desenvolver uma aplicação que permitisse ao usuário comparar todos os políticos candidatos à eleição de forma automática com base em suas ideologias e crenças políticas em relação às crenças individuais dos eleitores, utilizando como fonte dessa informação os votos dos políticos e dos eleitores em projetos reais.

As arquiteturas, ferramentas e metodologia de desenvolvimento atenderam às expectativas, permitindo que o desenvolvimento ocorresse de forma simples e organizada além de possibilitar a escalabilidade futura da aplicação sem maiores complicações. Todas as três principais funções da aplicação foram desenvolvidas com sucesso dentro do prazo estabelecido, permitindo que os eleitores tivessem acesso total à ferramenta projetada antes das eleições, e que assim pudessem votar de forma mais consciente e preparada, garantindo assim que suas crenças políticas fossem transferidas por meio de seus votos.

Como mostraram os testes com os usuários, os objetivos da aplicação foram alcançados, permitindo a realização de uma comparação fácil, rápida, precisa e abrangente entre os políticos do congresso e as ideologias individuais do usuário. Apesar de todos os objetivos terem sido alcançados, alguns tem mais espaço para melhora do que outros. Esse é o caso principalmente da abrangência da comparação, que poderia ser melhorada incluindo outros cargos do poder legislativo, como deputados estaduais e vereadores, assim como cargos do poder executivo.

Entretanto, o impacto positivo do trabalho baseia-se na premissa de que para melhorar a sociedade os governantes devem representar as pessoas e suas ideologias. Após a vitória de um candidato que sempre foi aberto quanto sua ideologia, levanta-se uma questão interessante.

Fazer com que os governantes representem as pessoas é bom?

6. Referências

Encyclopaedia Britannica, Inc. **Solon**. 2019. Disponível em https://www.britannica.com/biography/Solon. Acesso em 14/07/2019.

Aghaei, S. Nematbakhsh, M. A. Farsani, H. K. **Evolution of the World Wide Web: From Web 1.0 to Web 4.0**. International Journal of Web & Semantic Technology (IJWesT), Vol.3, No.1, January 2012.

Choudhury, N. **World Wide Web and Its Journey from Web 1.0 to Web 4.0**. International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 5, No.6. 2014.

Sommerville, I. **Engenharia de Software**. 8ª edição. Editora Pearson. Pág. 40-50, 262-266. 2007.

Schach, S. **Engenharia de Software: Os paradigmas clássico & orientado a objetos**. 7ª edição. Editora McGraw-Hill. Pág. 50-66. 2008.

Retirada de Sommerville, I. **Engenharia de Software**. 9ª edição. Editora Pearson. Pág. 60-68. 2011.

Oluwatosin, H. S. **Client-Server Model**. Journal of Computer Engineering (IOSR-JCE), Volume 16, Issue 1, Ver. IX, Pág. 67-71. Feb. 2014

Oliveira, H. E. M. **Aplicativo Cliente-Servidor multicamadas para controle de uma rede de lojas via WEB utilizando Java**. Departamento de Informática e Estatística - Universidade Federal de Santa Catarina (UFSC). Pág. 10-29. 2003.

Pressman, R S.. **Software Engineering: A Practitioner's Approach**. 7ª Edição. Pág. 384-392. (2010)

Framework7. **Introduction**. 2019. Disponível em https://framework7.io/docs/introduction.html. Acesso em 14/07/2019.

PHP. **O que é o PHP?**. 2019. Disponível em https://www.php.net/manual/pt_BR/intro-whatis.php. Acesso em 14/07/2019.

ArangoDB. **ArangoDB v3.4.6 Documentation**. 2019. Disponível em https://www.arangodb.com/docs/stable/. Acesso em 14/07/2019.