UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM ENGENHARIA DE SOFTWARE

OTÁVIO AUGUSTO DE SOUSA

ARQUITETURA RCWABH5-SOA PARA APLICAÇÕES RIA DESENVOLVIDAS EM HTML5

QUIXADÁ
2016

OTÁVIO AUGUSTO DE SOUSA

ARQUITETURA RCWABH5-SOA PARA APLICAÇÕES RIA DESENVOLVIDAS EM HTML5

Monografia apresentada ao Curso de Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Orientadora Profª. Antônia Diana Braga Nogueira

QUIXADÁ
2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

S696a Sousa, Otávio Augusto de
Arquitetura RCWABH5-SOA para aplicações RIA desenvolvidas em HTML5/ Otávio Augusto de Sousa. – 2016.
43 f.: il. color., enc.; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2016.
Orientação: Profª. Me. Antônia Diana Braga Nogueira
Coorientação: Prof. Me. Alisson Barbosa de Souza
Área de concentração: Computação

1. Arquitetura de Software 2. Aplicações Web 3. Software - desenvolvimento I. Título.

CDD 005.1

OTÁVIO AUGUSTO DE SOUSA

ARQUITETURA RCWABH5-SOA PARA APLICAÇÕES RIA DESENVOLVIDAS EM HTML5

Monografia apresentada ao Curso de Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.
Área de concentração: computação

Aprovado em: _____ / fevereiro / 2016.

BANCA EXAMINADORA

_____
Profª. MSc. Antônia Diana Braga Nogueira (Orientadora)
Universidade Federal do Ceará-UFC

_____
Prof. MSc. Alisson Barbosa de Souza (Coorientador)
Universidade Federal do Ceará-UFC

_____
Prof. MSc. Marcos Dantas Ortiz
Universidade Federal do Ceará-UFC

_____
Prof. Dr. Paulyne Matthews Jucá
Universidade Federal do Ceará-UFC

Aos meus pais, irmãos e a minha namorada ...

AGRADECIMENTOS

À Deus pelo dom da vida, sendo senhor e mestre sempre presente na minha vida, pois sem a presença de Deus na minha vida eu nada seria.

Aos meus pais e irmãos pelo apoio em todos os momentos da minha vida. Sendo verdadeiros gigante nos quais me apoiei, para chegar aqui.

À minha namorada por ter sido tão presente na elaboração desse trabalho, servindo de conselheira e amiga.

Aos meus orientadores que não mediram esforços para a realização deste trabalho.

Aos professores que passaram por minha vida estudantil, sendo mediadores do saber.

Aos meus colegas que durante esse período da graduação estiveram dispostos a me ajudar e contribuir para a minha formação. Com uma ênfase especial à Ana Cristina e ao Wagner que foram verdadeiros irmãos para mim.

"No fim tudo dá certo, e se não deu certo é porque ainda não chegou ao fim."
(Fernando Sabino)

RESUMO

Aplicações de internet rica, que são aplicações Web com características de aplicações desktop, vêm se popularizando cada dia mais. Criar arquiteturas que se adequam a esse tipo de aplicação é muito importante. Com base em estudos realizados, pôde ser constatado que HTML5 vem sendo uma das principais tecnologias no desenvolvimento desse tipo de aplicação. Este trabalho apresenta uma arquitetura projetada para aplicações de internet rica desenvolvidas em HTML5, que utiliza uma arquitetura orientada a serviços para o lado servidor. A arquitetura definida nesse trabalho foi validada a partir da criação de uma aplicação e da realização de testes em três cenários distintos. Os cenários na qual a aplicação foi avaliada foram: aplicação *online* e com banco de dados local, aplicação *online* sem banco de dados local e aplicação *off-line*. Na execução do trabalho, ficou clara a viabilidade da arquitetura, mas observou-se que os ganhos de desempenho proposto pala arquitetura depende do mecanismo de sincronização utilizado.

Palavras-chave: Arquitetura de Software. Aplicações Web. Desenvolvimento de Software.

ABSTRACT

Rich internet applications that are Web applications with features of desktop applications, has become more popular every day. Create architectures that suit this type of application is very important. Based on studies it could be see that HTML5 has been one of the key technologies in developing this type of application. This paper presents an architecture designed for rich Internet applications developed in HTML5, which uses a Service-Oriented Architecture for the server side. The architecture defined in this work was validated through the creation of an application and testing of three different scenarios. The scenarios in which the application was evaluated: online application with local database, online application without local database and offline application. In the running of the work, it became clear the viability of the architecture, but it was observed that the performance gains offered by the architecture depends on the synchronization mechanism used.

Keywords: Software Architecture. Web Applications. Software Development.

LISTA DE ILUSTRAÇÕES

LISTA DE GRÁFICOS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

O mundo está movendo-se dos desktops para a internet móvel (CHEN; LIU, 2012). Criar aplicações, que sejam compatíveis com a grande gama de dispositivos, com alto desempenho, uma boa usabilidade e baixo acoplamento entre os módulos é um desafio que os engenheiros de software vêm enfrentando frequentemente.

Aplicações Web convencionais estão longe de alcançar todos esses atributos de qualidade. Essas aplicações têm seu processamento centrado no servidor e, a cada interação do usuário, a página necessita ser atualizada, afetando de forma negativa a usabilidade e o desempenho dos sistemas. Por outro lado, as aplicações do tipo *Rich Internet Applications* (RIA) trazem parte do processamento para o cliente e diminuem a quantidade de atualizações das páginas Web (BUSCH; KOCH, 2009).

RIA são aplicações Web com características de aplicações *desktop*, que fornecem interfaces gráficas sofisticadas e com maior usabilidade, além de terem grande parte do processamento realizado no cliente. Segundo Pietruszkiewicz e Dzega (2009), as aplicações RIA também têm vantagens sobre as aplicações *desktop* por serem de fácil instalação, seguras, independentes de hardware ou plataformas e fáceis de atualizar.

Para desenvolver aplicações RIA, surgiram várias tecnologias, como mostrado em Pina e Oliveira (2013). No trabalho, foi realizado um estudo sobre os principais expoentes da RIA, que segundo os autores são: Adobe Flash, Microsoft Silverlight, Oracle JavaFX e HTML5. Os autores fizeram uma revisão sistemática através da seleção de artigos e publicações, que teve como conclusão evidências do forte crescimento do HTML5 em detrimento das demais tecnologias.

HTML (*Hypertext Markup Language*) é uma linguagem de marcação de hipertexto, aberta, mantida e desenvolvida pelo grupo de trabalho *World Wide Web Consortium* (W3C). HTML5 diz respeito à quinta versão dessa linguagem, que se diferencia das outras versões da HTML por encapsular os recursos oferecidos pela Web, como por exemplo: transmissão de áudio e vídeo, afim de evitar o uso de *plug-ins* (SANTINI, 2014).

Chen e Liu (2012) propõem a arquitetura *Rich Client Web Architecture based on HTML5* (RCWABH5) para aplicações construídas em HTML5, que possibilita a criação de

aplicações Web personalizáveis que podem ser utilizadas *off-line*, não necessitam de *plug-in* e realizam uma menor quantidade de requisições ao servidor.

A arquitetura RCWABH5 provê ganhos de desempenho no cliente em relação às arquiteturas cliente-servidor tradicionais. No entanto, a mesma não faz uma adaptação da arquitetura do lado servidor, que continua com uma arquitetura *browser*-servidor tradicional. Com isso os ganhos de desempenho e a diminuição do uso dos recursos computacionais no lado servidor não são maximizados da forma como seria se estivesse usando uma arquitetura adaptada para aplicações RIA.

Segundo a Infosys (2007), RIA é a melhor solução para se trabalhar com as aplicações que utilizam Arquitetura Orientada a Serviço (SOA, do inglês *Service-Oriented Architecture*), um paradigma arquitetural para organizar e gerenciar os recursos empresariais provendo um baixo acoplamento entre os serviços que regem o processo de negócio (INFOSYS, 2007). Cada vez mais, as empresas estão migrando para uma arquitetura orientada a serviço para alinhar o setor de tecnologia da informação (TI) e seus processos de negócio.

O objetivo geral desse trabalho consistiu em aprimorar a arquitetura RCWABH5, definindo uma arquitetura baseada em SOA para o lado servidor, chamada RCWABH5-SOA. Os objetivos específicos deste trabalho são: definir uma arquitetura utilizando SOA para o lado servidor da arquitetura RCWABH5; criar uma aplicação RIA desenvolvida em HTML5 que implementa a arquitetura RCWABH5-SOA; medir o desempenho da aplicação RIA criada utilizando diferentes cenários e fazer análise da execução da aplicação que implementa a arquitetura RCWABH5-SOA.

Este trabalho está organizado da seguinte forma: na Seção 2, apresenta-se trabalhos relacionados; na Seção 3, está a fundamentação teórica; na Seção 4, os procedimentos metodológicos; na Seção 5, é apresentada a arquitetura RCWABH5-SOA; na Seção 6, está o desenvolvimento e na Seção 7, são apresentadas as considerações finais acerca do trabalho realizado e propostas de trabalhos futuros.

2 TRABALHOS RELACIONADOS

Chen e Liu (2012) propõem uma arquitetura para aplicações RIA desenvolvidas em HTML5, chamada de RCWABH5. Na arquitetura definida por Chen e Liu (2012), foi utilizada a arquitetura Browser/Server (B/S), definida uma arquitetura para o lado cliente e acrescentada

uma camada entre o navegador do cliente e o servidor. Essa camada é chamada de *client control*, sendo a chave para RIA em RCWABH5.

Em Santini (2014), foi realizado um estudo sobre o desenvolvimento de aplicações RIA com JavaFX, em que foram avaliadas as principais características de JavaFX através de um estudo de caso e comparação com outras tecnologias semelhantes. Para o estudo de caso, Santini desenvolveu uma aplicação RIA com JavaFX.

Guo et al. (2010) apresenta o estudo da arquitetura da aplicação *Geographic Information System* (GIS), uma arquitetura baseada em SOA e RIA. GIS consiste em uma aplicação que junta diferentes serviços de mapas, afim de agregar e mostrar dados de diferentes domínios. A comunicação com os serviços externos se dá através de SOAP ou REST, essa comunicação é feita no lado servidor da aplicação. Por outro lado, a comunicação entre RIA e SOA é feita através de *FLEX Remoting*.

Zhang (2010) apresenta uma arquitetura de duas camadas para aplicações na "nuvem" com RIA maximizado e SimpleBD através de REST minimizado. O trabalho de Zhang (2010) apresenta uma arquitetura que foca o processamento das regras de negócio no lado cliente e usa o lado servidor apenas para armazenar os dados. Como mecanismo de comunicação entre cliente e servidor, a arquitetura utiliza um REST minimizado.

O Quadro 1 mostra uma comparação entre os trabalhos relacionados e o proposto neste trabalho.

Quadro 1- Comparação entre os trabalhos relacionados e o trabalho proposto

Trabalho	Semelhança	Diferença
Chen e Liu	Definição de uma arquitetura baseada em HTML5 para aplicações RIA.	Chen e Liu não definem uma arquitetura para ser utilizado no lado servidor. Este trabalho apresentada uma arquitetura baseada em SOA para ser utilizado no lado servidor.
Santini	Desenvolvimento de uma aplicação RIA	Santini utiliza JavaFX no desenvolvimento da aplicação RIA. Neste trabalho foi utilizado HTML5.
Guo	Apresenta uma arquitetura para aplicações RIA que utiliza SOA para o lado servidor.	E diferente deste trabalho, os serviços externos são acessados no lado servidor.
Zhang	Utiliza REST como mecanismos de comunicação entre cliente e servidor	O trabalho proposto usa SOA no lado servidor. Zhang utiliza para o lado servidor Amazon SimpleBD, um *Web Service* para rodar consultas em dados estruturados.

Fonte: Elaborada pelo autor

3 FUNDAMENTAÇÃO TEÓRICA

3.1 *Rich Internet Applications* (RIA)

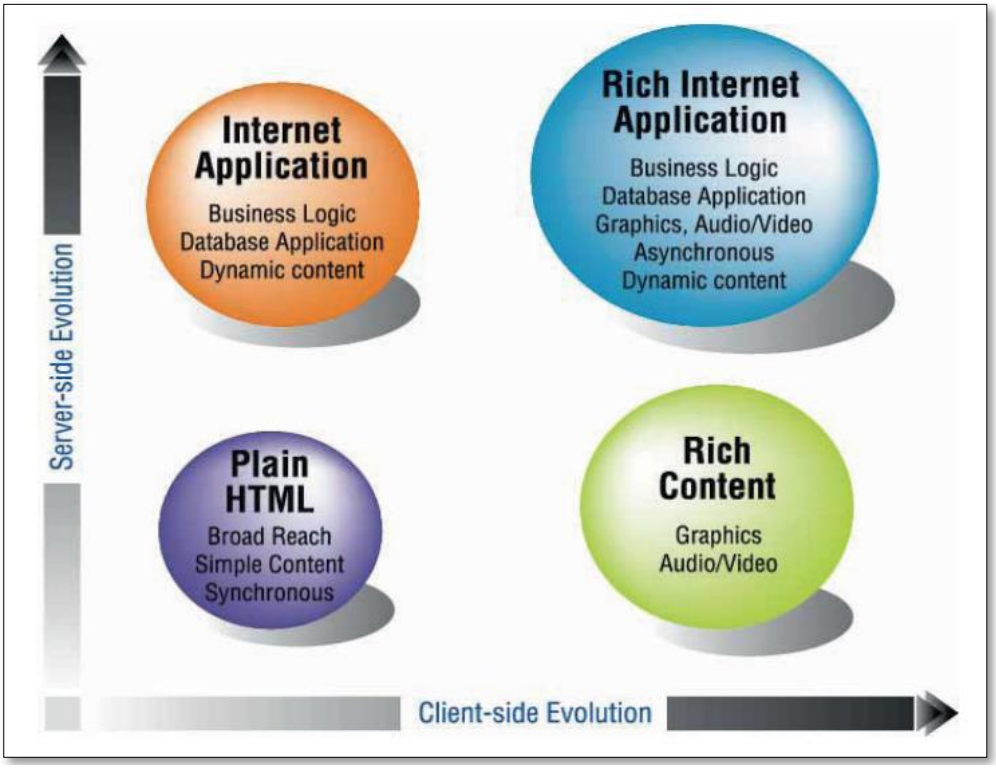
As aplicações RIA são uma evolução das aplicações Web convencionais, que usavam páginas HTML estáticas, tecnologia que originalmente foi projetada para entregar documentos com informações de forma síncrona via navegador. Mas, com o passar do tempo,

surgiu a necessidade de atualizar os dados de forma assíncrona e aumentar a usabilidade (INFOSYS, 2007).

Até o surgimento das aplicações RIA, a atualização dos dados consistia em recarregar as páginas Web, o que resultava em lentidão e uma baixa usabilidade. Além da incapacidade de representar visualmente dados grandes e complexos, essas aplicações não davam suporte à manipulação dos elementos de forma interativa para o usuário (INFOSYS, 2007).

Na Figura 1, é apresentando um quadrante da evolução das aplicações Web. É possível identificar RIA como sendo o tipo de aplicação com maior evolução neste cenário. Na vertical, é representada a evolução das aplicações no lado servidor e, na horizontal, representada a evolução no lado cliente. Os círculos representam os tipos de aplicações Web e as áreas que elas atendem.

Figura 1- Evolução das aplicações Web



Fonte: INFOSYS (2007)

Aplicações RIA, segundo Pina e Oliveira (2013), são aplicações Web com características de aplicações *desktop*. Essas aplicações oferecem interface gráfica sofisticada, intuitivas e ricas em animação e recursos multimídia.

Para Busch e Koch (2009), RIA são aplicações Web em que os dados podem ser processados tanto no cliente quanto no servidor. Esses dados são enviados assincronamente

entre o cliente e o servidor. No cliente, RIA provê um visual parecido com aplicações desktop e a palavra "*rich*" significa particularmente a diferença da geração anterior das aplicações Web, onde as mesmas não tinha recursos como: banco de dados local, áudio/vídeo e comunicação assíncrona.

Esse tipo de aplicação está ficando cada vez mais popular e uma ampla gama de tecnologias podem ser utilizadas para criar essas aplicações, como: Adobe Flash, Microsoft Silverlight, Oracle JavaFX e HTML5. Uma visão mais detalhada sobre cada tecnologia pode ser encontrada em Pina e Oliveira (2013).

No presente trabalho, é utilizada a definição de RIA utilizada por Bush e Kosh (2009) e, para o desenvolvimento da arquitetura, é definida uma arquitetura para aplicações RIA baseadas em HTML5.

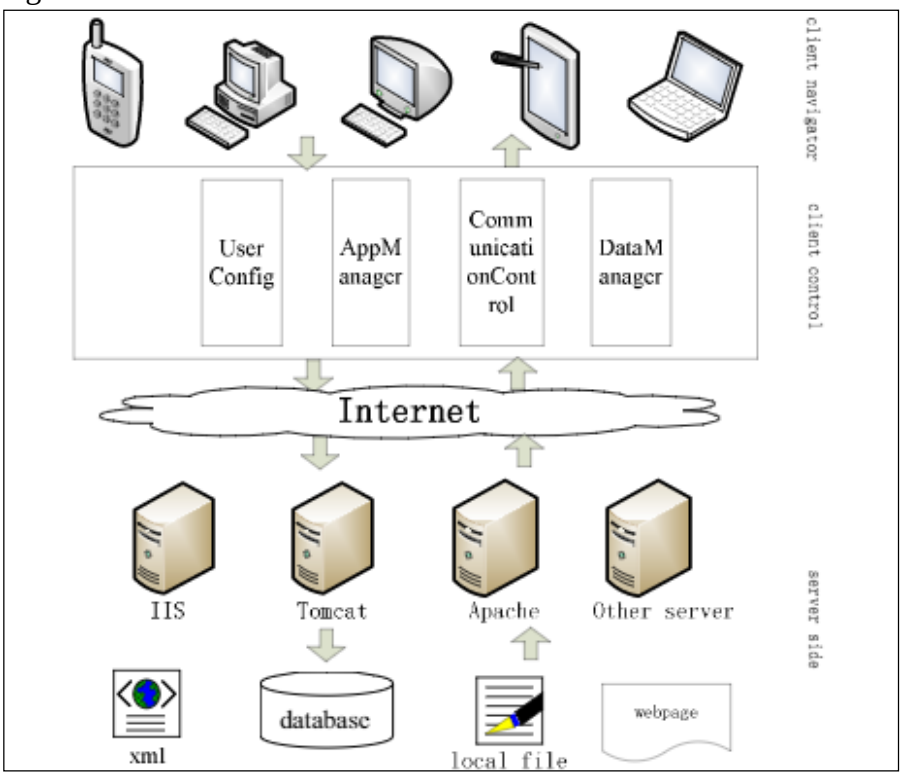
3.2 Rich Client Web Architecture Based on HTML5 (RCWABH5)

RCWABH5 é uma arquitetura para aplicações ricas baseadas em HTML5, que foi proposta por Chen e Liu (2012). Essa arquitetura tem como objetivo reduzir a quantidade de requisições ao servidor, armazenar dados no cliente para reduzir a quantidade de conexões entre o cliente e o servidor e possibilitar o uso da aplicação mesmo que ela fique *off-line*, sem a necessidade de instalação um *plug-in* (CHEN; LIU, 2012).

A arquitetura RCWABH5 usa uma arquitetura *browser*/servidor tradicional para o lado servidor e define a camada *client control* para fazer o controle da aplicação no lado cliente e usa o protocolo HTTP para comunicação entre o cliente e o servidor.

Na Figura 2, é apresentada a estrutura da RCWABH5 e suas três camadas: *client navigator*, *client control* e *server side*.

Figura 2 - Estrutura RCWABH5

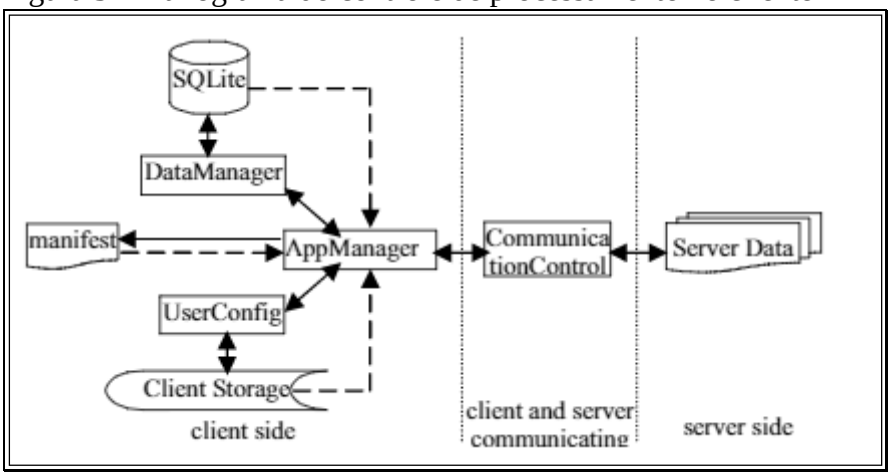


Fonte: CHEN e LIU (2012)

Na camada *client navigator*, temos os navegadores, programas que habilitam os usuários a interagirem com os documentos HTML das aplicações Web. Os navegadores podem ser de diferentes tipos e empresas, possibilitando assim uma maior compatibilidade entre as aplicações RIA e os dispositivos do usuário.

A camada *client control* é a principal camada da arquitetura RCWABH5, diferenciando a arquitetura RCWABH5 das arquiteturas cliente-servidor convencionais. Nesta camada, temos quatro módulos responsáveis por fazer o controle da aplicação que são: *AppManager*, *ComunicationControl, DataManager e UserConfig*. Na Figura 3, temos de forma mais detalhada como esses módulos se comunicam.

Figura 3 - Fluxograma do controle de processamento no cliente



Fonte: CHEN e LIU (2012)

AppManager é o módulo responsável por definir quais dados serão processados no cliente ou no servidor de acordo com a rede e o tipo de requisição. O módulo *ComunictaionControl* processa a comunicação entre o *browser* e o servidor quando a aplicação está *online.*

Já o módulo *DataManager* é responsável por manter os dados no cliente atualizados e quando a aplicação estiver *off-line* armazena as atualizações no cliente e quando a conexão for reestabelecida enviar os dados para o servidor. O *UserConfig* é responsável por armazenar e gerenciar os dados referentes a configurações de interface realizada pelo usuário.

A camada *server side* na arquitetura RCWABH5, utiliza uma arquitetura de *Web server* convencional, que hospeda as páginas Web e armazena os dados. A arquitetura RCWABH5 foi utilizada neste trabalho como base para a definição da arquitetura RCWABH5-SOA que define uma arquitetura SOA para o lado servidor e utiliza a arquitetura no lado cliente que foi definida na arquitetura RCWABH5. Desse modo, as camadas e módulos definidos por Chen e Lui (2012) foram implementas no lado cliente da aplicação de teste conforme descrito no trabalho.

3.3 Arquitetura orientada a serviços (SOA, do inglês *Service-Oriented Architecture*)

Arquitetura orientada a serviços são formas de desenvolvimento de sistemas distribuídos onde os componentes são sistemas autônomos, executados em computadores geograficamente distribuídos (SOMMERVILLE, 2011). Um conjunto de protocolos-padrões foram estabelecidos para fornecer suporte à comunicação dos serviços e à troca de informação.

Elr (2009) define SOA como um modelo arquitetônico que visa aprimorar a eficiência, a agilidade e a produtividade de uma empresa e tem os serviços como suporte na realização dos objetivos estratégicos associados à computação orientada a serviços.

Para Josuttis (2008), SOA é um paradigma arquitetural para lidar com os processos corporativos distribuídos em grandes sistemas heterogêneos existentes e novos que estão sob controle de diferentes proprietários. SOA tem como conceito técnico-chave: serviços, interoperabilidade e acoplamento fraco.

Segundo Josuttis (2008), serviço é uma interface para mensagens que são trocadas entre fornecedor e consumidor. Do ponto de vista de negócio, serviço esconde os detalhes técnicos e permite que as pessoas de negócio lidem com ele.

A interoperabilidade é um requisito fundamental de SOA, pois é a capacidade de usar diferentes plataformas e produtos para implementar os serviços. E acoplamento fraco é o conceito empregado para lidar com os requisitos de escalabilidade, flexibilidade e tolerância à falhas, tendo o objetivo de minimizar as dependências (JOSUTTIS; 2008).

A utilização de SOA permite que sistemas desenvolvidos em diferentes linguagens e de empresas diferentes possam se comunicar e trocar informações. Essa característica de SOA possibilita uma maior automação dos processos de negócio, além de possibilitar modificações no sistema sem causar um impacto tão grande.

Neste trabalho, os conceitos de SOA são utilizados na definição da arquitetura do lado servidor por facilitar a modularização e integração com possíveis sistemas já existentes. São utilizados Web Services para realizar a infraestrutura de SOA, onde o protocolo utilizado é o HTTP.

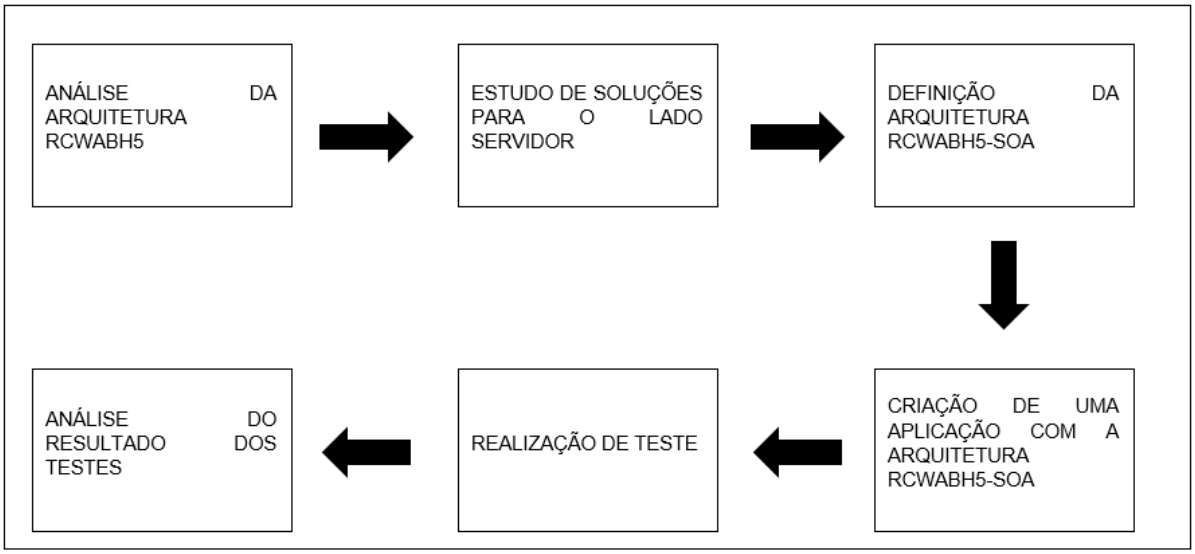
4 PROCEDIMENTOS METODOLÓGICOS

Com o objetivo de aprimorar a arquitetura RCWABH5, foi definida uma arquitetura baseada em SOA para o lado servidor. Desse modo, foi realizada a definição da arquitetura RCWABH5-SOA, onde foi elaborado o projeto da arquitetura (ver Seção 5).

Após a definição da arquitetura, foi definida e desenvolvida uma aplicação para validar a arquitetura e verificar se a mesma possuía condições técnicas de ser utilizada em aplicações RIA. Com a aplicação finalizada, foram realizados testes para medir o desempenho da arquitetura. Após o processo de teste finalizado, foram realizadas análises sobre os dados coletados.

Na Figura 4, são apresentados os procedimentos metodológicos que foram utilizados neste trabalho. Nas seções abaixo, é apresentado o detalhamento dos passos ilustrados na figura.

Figura 4 - Procedimentos Metodológicos



Fonte: Elaborada pelo autor

4.1 Análise da arquitetura RCWABH5

Afim de compreender as carências da arquitetura RCWABH5, foi realizado uma análise da mesma com o propósito de identificar os pontos de melhoria. O resultado da análise foi que a mesma não tinha uma arquitetura para o lado servidor bem definida, não ficando claro o que deveria ser utilizado e como deveria ser feito. Esta decisão ficava a cargo de quem iria usar a arquitetura.

Por conta da falta de dados sobre a arquitetura RCWABH5, foi necessário definir uma arquitetura para o lado servidor. Essa arquitetura deve ser compatível com a arquitetura do lado cliente. Além disso, era desejado obter benefícios das aplicações RIA, como o processamento no cliente e o funcionamento *off-line*.

4.2 Estudo de soluções para o lado servidor

Com a intenção de melhorar a arquitetura e atender requisitos arquiteturais como: escalabilidade, manutenibilidade e desempenho, foi realizado um levantamento das possíveis arquiteturas que poderiam ser utilizadas. Observou-se a existência de vários estilos arquiteturais que poderiam ser utilizados na criação de uma arquitetura para o lado servidor.

Uma das soluções seria a arquitetura cliente/servidor, que divide o sistema em duas aplicações, onde o cliente faz requisições ao servidor. Nesse tipo de arquitetura, as aplicações cliente e servidor ficam totalmente acopladas, prejudicando a escalabilidade e manutenibilidade (MICROSOFT, 2016).

Outro estilo que poderia ser utilizado é a *Message Bus*. Esse tipo arquitetural consiste em aplicações que utilizam mensagens para se comunicar, onde uma aplicação não precisa saber detalhes de implantação da outra. Podem ser utilizados diferentes canais de comunicação entre as aplicações (MICROSOFT, 2016).

Dentre os diversos estilos arquiteturais analisados, SOA é o que mais se adequa ao conceito de aplicação RIA (INFOSYS, 2007) por possibilitar que a aplicação cliente seja constituída com tecnologias diferentes das tecnologias utilizadas no desenvolvimento dos serviços. Além de atender todos os requisitos arquiteturais definidos para a arquitetura do lado servidor.

4.3 Definição da arquitetura RCWABH5-SOA

Nesta etapa, foi desenvolvido o projeto da arquitetura RCWABH5-SOA, onde foram definidos os módulos da arquitetura e o processo de comunicação entre eles. Algumas visões foram criadas e os lados cliente e servidor da arquitetura foram definidos (ver Seção 5). Também foi realizado um detalhamento da função dos módulos, afim de tornar a arquitetura mais completa.

Foi necessário realizar modificações no módulo *Communication Control,* com a criação de um módulo, chamado de *Syncronization Manager,* responsável pela sincronização dos dados entre o cliente e o servidor.

4.4 Criar uma aplicação com a arquitetura RCWABH5-SOA

A etapa inicial da criação de uma aplicação consiste em sua concepção e elicitação dos requisitos. Foi feita a definição da aplicação que será usada como estudo de caso e foi aplicada a arquitetura RCWABH5-SOA. Foi realizado um *brainstorm* para a definição da aplicação que foi desenvolvida. No *brainstorm*, foi decidido o desenvolvimento de uma aplicação para o gerenciamento de *backlogs* em projetos de software, chamada ScrumTool.

4.5 Realização de Testes

Com a aplicação devidamente construída com base na arquitetura RCWABH5-SOA, foram feitos testes e realizou-se um conjunto de medições para verificar o desempenho da aplicação. Foram coletados dados sobre a quantidade de *upload* e *download* de dados, quantidade de requisições efetuadas e tempo de conexão gasto. Para avaliar a arquitetura proposta, foram criados também três diferentes cenários: aplicação *online* e com banco de dados local, aplicação *online* sem banco de dados local e aplicação *off-line*.

4.6 Análise dos resultados dos Testes

Com os dados sobre os testes, foi realizada uma análise do desempenho da aplicação que implementou a arquitetura RCWABH5-SOA. A análise levou em consideração a diferença dos resultados nos três cenários que foram testados e possibilitou a inclusão de trabalhos futuros complementares a este trabalho.

5 ARQUITETURA RCWABH5-SOA

A arquitetura RCWABH5-SOA baseia-se na arquitetura RCWABH5, que foi previamente apresentada neste trabalho. A arquitetura RCWABH5-SOA diferencia-se da arquitetura base por adicionar uma arquitetura baseada em SOA para o *lado* servidor da aplicação, tornando-se uma arquitetura RCWABH5-SOA menos genérica e mais completa. A arquitetura proposta visa atender requisitos arquiteturais como: escalabilidade, manutenibilidade e desempenho.

A escolha por SOA para o lado servidor se deu por suas características que aumentam a escalabilidade e manutenibilidade de um sistema. O fato de ter as funcionalidades divididas em serviços separados que podem ser modificados e/ou escalados de forma a não ter impacto nos outros serviços, torna esse tipo de arquitetura muito utilizada no mercado (INFOSYS, 2007).

Para o requisito de desempenho, optou-se por utilizar o cliente descrito na arquitetura RCWABH5, onde tem-se um cliente rico que pode executar várias operações com poucas interações com um servidor. Além disso, permite o funcionamento da aplicação mesmo o usuário estando *off-line.* O funcionamento *off-line* deve ser uma escolha de projeto, tendo em vista que algumas funcionalidades só devem ser permitidas quando o mesmo estiver conectado à Web.

A possibilidade de funcionamento *off-line* da aplicação torna a experiência do usuário mais prazerosa, tendo em vista que o usuário pode continuar utilizado os sistemas quando a conexão com a internet não estiver disponível. Desse modo, a conexão deixa de ser um empecilho para o usuário executar suas tarefas.

Um exemplo de ferramenta que pode ganhar proveito dessa característica de funcionamento *off-line* é um sistema de controle de ponto, onde o horário de entrada e saída podem ser registrados no sistema mesmo na ausência de conexão. Neste caso, quando a conexão for reestabelecida as informações referentes aos pontos serão enviadas para o servidor.

A arquitetura RCWABH5-SOA divide-se em duas grandes partes: o cliente e o servidor. O cliente é uma aplicação RIA que executa seu processamento no navegador do usuário e se comunica com os serviços através da *Web*. Os serviços dessa arquitetura são o que compõem o lado servidor. Além dos serviços, o lado servidor possui um servidor *Web* que hospeda aplicação.

Há diferentes tipos de serviços que podem ser usados por aplicações que utilizam essa arquitetura. Podem ser usados serviços interno, externos, sincronizáveis e não sincronizáveis. Os serviços internos são os serviços que estão sobre o mesmo domínio da aplicação. Por exemplo, a aplicação está sob o domínio www.exemplo.com:80 e o serviço está sob o domínio www.exemplo.com:8081/servico. Serviços externos são serviços que estão sobre outro domínio e/ou são fornecidos por outras empresas.

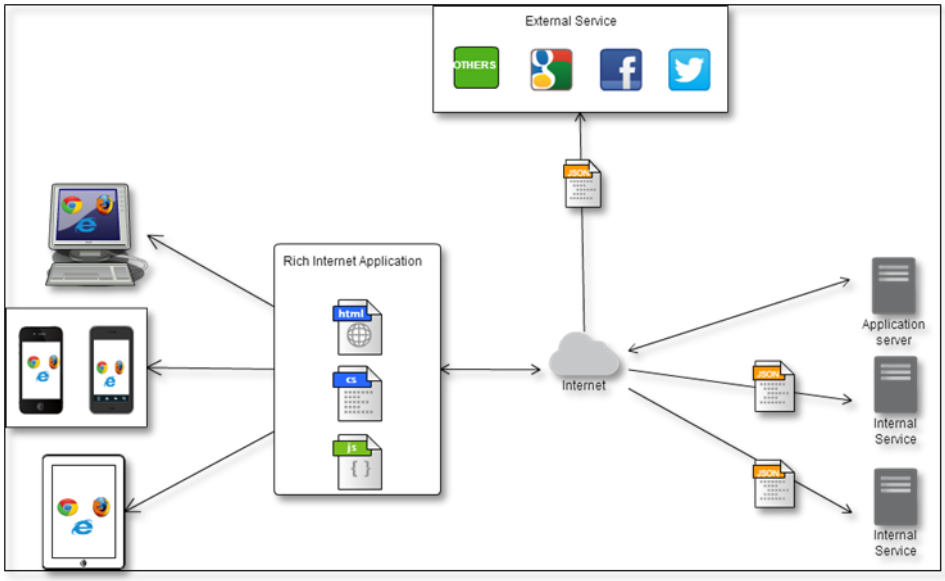
Serviços sincronizáveis são os serviços que têm um mecanismo de sincronização com as aplicações cliente. Esses serviços enviam notificações para a aplicação RIA alertando que modificações foram feitas e a aplicação RIA vai se encarregar de buscar essas modificações ou o serviço manda uma mensagem contendo as modificações que ocorreram. Sincronização é essencial para aplicações em tempo real e para ferramenta de colaboração simultânea. Os serviços não sincronizáveis são aqueles que não possuem esse mecanismo ficando a cargo da aplicação cliente buscar as modificações.

A troca de informações entre a aplicação cliente e os serviços deve se dar através de arquivos em formato JSON (Notação de Objetos JavaScript, do inglês *JavaScript Object Notation*), por ser em formato texto, completamente independente de linguagem e através da Web.

Uma visão geral da arquitetura pode ser vista na Figura 5. Os serviços externos são estão representados no quadro *External Service*, onde tem o exemplo de alguns serviços

externos como Google, Facebook e Twitter. Os serviços internos estão representados pelos *internal services*, que se comunicam com a aplicação RIA através da internet utilizando REST e JSON. A aplicação RIA, está representada pelo retângulo *Rich Internet Application*, que contém os arquivos que serão executados no navegador do cliente, que está representado pela as plataformas móveis e *desktop.*

Figura 5 - Arquitetura RCWABH5-SOA: Uma visão geral



Fonte: Elaborada pelo autor

Na Figura 6, são apresentados os módulos que compõem a arquitetura RCWABH5-SOA. O *client-side* é composto pelos módulos *Communications Control*, *App Manager*, *Data Manager* e *User Config*.

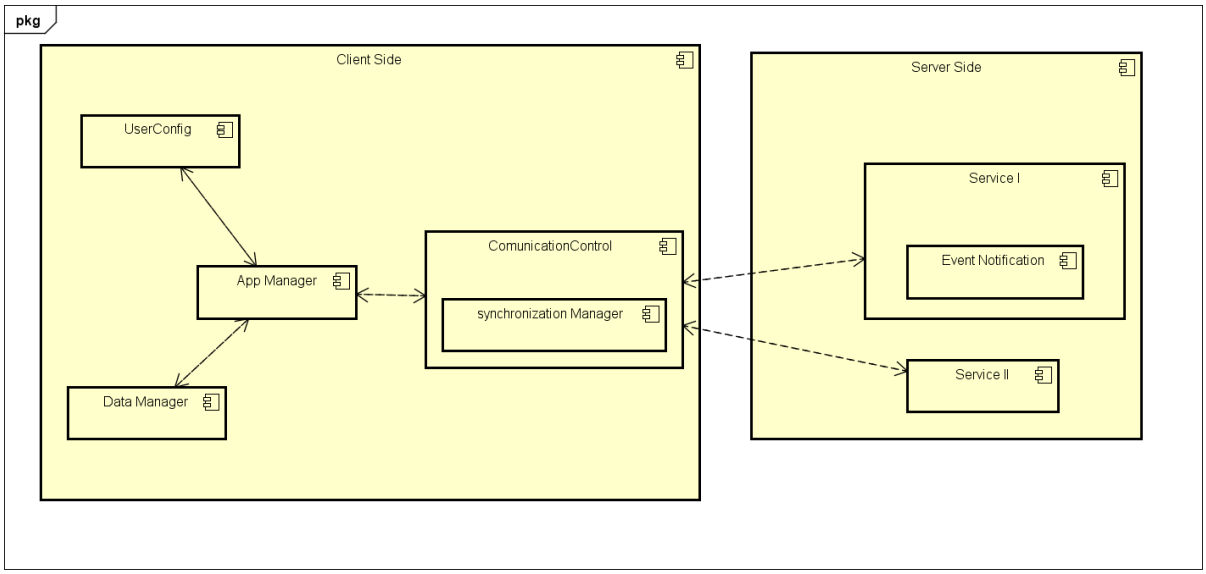
Communication Control é o módulo responsável pela comunicação entre a aplicação RIA e os serviços. É responsabilidade do *Communication Control* o gerenciamento do estado da rede, conexão e chamadas dos serviços. Para a sincronização dos dados entre a aplicação RIA e os serviços, a arquitetura RCWABH5-SOA define um sub módulo do *Communication Control*, chamado *Syncronization Manager*.

App Manager é responsável pela integração entre os demais módulos da aplicação RIA, processamento e apresentação dos dados e regras de negócio da aplicação.

User Config é o módulo responsável pelo gerenciamento das configurações feita pelo usuário. Por exemplo, se a aplicação permite a criação de temas e cores personalizáveis, o *User Config* irá armazenar e gerenciar essas configurações.

Data Manager tem a responsabilidade de gerenciar o banco de dados local que é fornecido pelo navegador. Ficando a cargo desse módulo conexão, criação das estruturas de armazenamento e gerenciamento dos dados.

Figura 6 - Módulos que compõem a arquitetura RCHWABH5-SOA



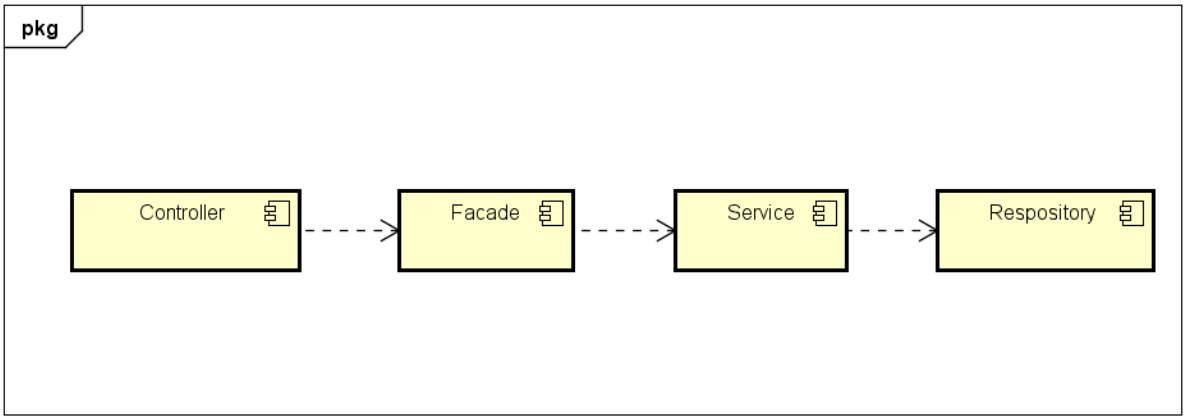
Fonte: Elaborada pelo autor

O *server side* é composto pelos serviços que podem ou não ter um mecanismo de sincronização como foi explicado anteriormente. Uma visão básica de um serviço pode ser vista na Figura 7. No entanto, a organização dos componentes pode mudar de acordo com a necessidade de cada projeto.

A estrutura básica de um serviço, proposta pela arquitetura RCWABH5-SOA, consiste das seguintes camadas:

- *Controller* é uma interface REST para o serviço. Nesse módulo que são criadas e definidas as chamadas REST.
- *Facade* é a camada responsável por encapsular as funcionalidades do serviço, impedindo que a camada *controller* tenha acesso direto a tais funcionalidades.
- *Service* é a camada responsável pela aplicação das regras de negócio do serviço, validações dos dados e comunicação com a camada *repositor*.
- *Repository* é camada responsável pela persistência dos dados.

Figura 7 - Diagrama de pacote básico de um serviço



Fonte: Elaborada pelo autor

6 APLICAÇÃO SCRUMTOOL

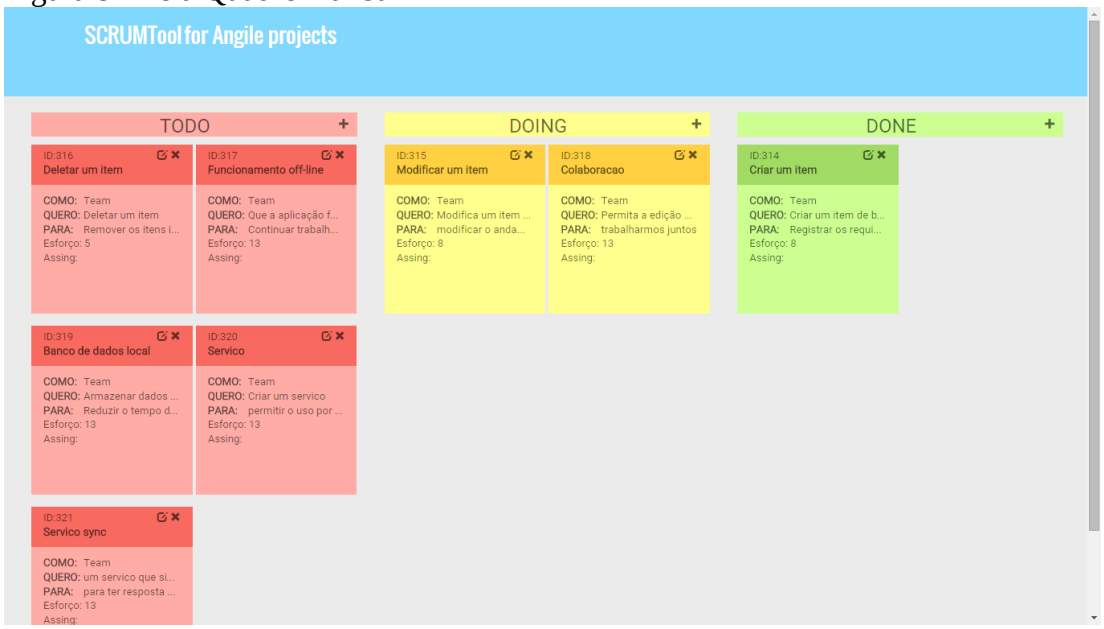
A fim de testar e analisar os resultados da arquitetura RCWABH5-SOA, foi criada uma aplicação chamada ScrumTool, uma ferramenta para uso em projetos de software onde uma equipe de desenvolvimento pode fazer o gerenciamento do *backlog* do projeto de forma colaborativa. A aplicação possibilita a criação de itens com histórias de usuário para o *backlog* e a organização desses itens em um quadro *kanban*, onde pode ser verificado o andamento dos itens no projeto, como pode ser observado na Figura 8.

Na aplicação ScrumTool, foram utilizadas funcionalidades do HTML5 que tornam a aplicação mais rica, como por exemplo o uso de *drag and drop*[1], *IndexedDB*[2], *Server-Sent Events*[3]. O *drag and drop*, foi utilizada na atualização do status de um item. *IndexedDB*, foi utilizado para armazenamento dos dados localmente. *Server-Sent Events*, foi utilizado no mecanismo de sincronização, para realizar a sincronização dos dados em tempo real.

O desenvolvimento da aplicação foi dividido em dois projetos: um projeto para criar o lado cliente da aplicação e o outro para criar um serviço. Foi utilizado o Git[4] como sistema de gerenciamento de versão (o código da aplicação pode ser encontrado em: https://github.com/OtavioAugustoSousa/ScrumTool - ver Apêndice A). Uma descrição detalhada do cliente e do serviço será apresentado no decorrer da seção.

[1] http://www.w3schools.com/html/html5_draganddrop.asp
[2] www.w3.org/TR/IndexedDB
[3] http://www.w3schools.com/html/html5_serversentevents.asp
[4] https://git-scm.com/

Figura 8 - Tela Quadro Kanban



Fonte: Elaborada pelo autor

6.1 Aplicação cliente

A aplicação cliente corresponde ao *client-side* da arquitetura RCWABH5-SOA, uma aplicação RIA que é executada no navegador do usuário. No desenvolvimento da aplicação foram utilizadas as seguintes tecnologias: HTML5, JavaScript com o *framework* AngularJS, IndexedDB, Bootstrap. As tecnologias utilizadas no ambiente de desenvolvimento podem ser vistas no Quadro 2.

a) AngularJS

AngularJS[5] é um framework para a linguagem JavaScript muito utilizado no mercado e é mantido por engenheiros da Google[6]. Sua escolha se deu principalmente pelo fato de se encaixar perfeitamente nos conceitos do *client-side* da arquitetura RCWABH5-SOA. A versão utilizada do framework foi AngularJS v1.4.8.

b) Bootstrap

Bootstrap[7] é um framework HTML, CSS e JavaScript, que contém uma grande quantidade de componentes prontos e permite a criação de sistemas responsivos, que são sistemas que se adaptam ao tamanho da tela, de forma simples. O uso de Bootstrap se deu na criação do layout da aplicação e na utilização de alguns de seus componentes.

[5] www.angularjs.org
[6] www.google.com
[7] www.getbootstrap.com

c) IndexedDB

IndexedDB[8] é uma API (Interface de Programação de Aplicação, *Application Programming Interface*) para armazenamento de dados no navegador do usuário, permitindo o armazenamento de uma grande quantidade de dados. O uso de IndexedBD se deu pela necessidade de funcionamento *off-line* da aplicação, para isso se fez necessário armazenar dados no navegador do usuário.

Quadro 2- Ferramentas utilizadas no desenvolvimento da aplicação do lado cliente

Nome	Tipo	Versão	Descrição
WebStorm	IDE	11	Ferramenta utilizada na edição e codificação da aplicação do lado cliente.
Google Chrome	Navegador	48.0.2564.109 m	Navegador Web, utilizado para executar a aplicação cliente
Git	Sistema de controle de mudanças	2.7.1	Utilizado para gerenciar as mudanças da aplicação.
Windows 10	Sistema Operacional	Home Single Language	Sistema Operacional utilizado para executar as ferramentas de desenvolvimento.

Fonte: Elaborada pelo autor

[8] www.w3.org/TR/IndexedDB

6.2 Lado servidor

O projeto do lado servidor foi baseado na arquitetura RCWABH5-SOA, onde a arquitetura prevê a criação de uma aplicação baseada em SOA para o lado servidor. Essa aplicação constituiu-se basicamente de um serviço para armazenar os dados dos itens do *backlog* e um serviço de notificações de mudanças que foi utilizado na sincronização.

Para construir essa aplicação, foi criado um *Web Service* REST (Transferência de Estado Representacional, do inglês *REpresentational State Transfer*), que é uma das formas de implementar SOA, que foi escolhido por sua simplicidade e por atender os requisitos da arquitetura.

Na construção do *Web Service* da aplicação, foi utilizada a linguagem Java e o *framework* Spring[9]. A escolha da linguagem e do *framework* foi baseada na sua popularidade, utilização no mercado e por sua simplicidade e facilidade para criar um *Web Service*.

Uma visão de como ficou a estrutura dos pacotes e as chamadas REST do serviço pode ser observada respectivamente, na Figura 9 e no Quadro 3. No Quadro 4, estão as tecnologias utilizadas no ambiente de desenvolvimento dos serviços.

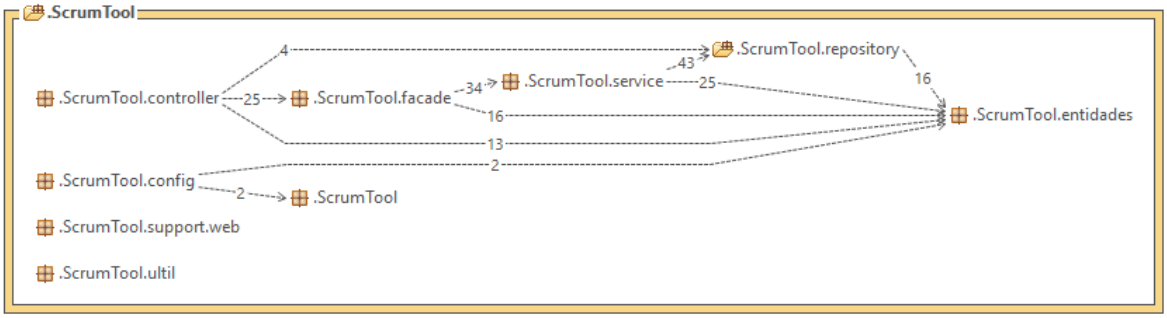
Na Figura 9, pode ser observado claramente que a estrutura proposta na arquitetura RCWABH5-SOA foi seguida, mas necessitou da criação de alguns pacotes extras por conta da utilização do *framework* Spring.

No Quadro 3, está representada as chamadas criadas para o serviço que foi criado para armazenar os dados do *backlog*. Onde a primeira coluna diz respeito aos métodos HTTP que devem ser usado para a execução da funcionalidade desejada, de acordo com o padrão definido na arquitetura REST. Onde o método POST deve ser utilizado para persistência de dados, o método GET deve utilizado para recuperar informações do servidor, o método PUT deve ser usado quando for fazer atualização dos dados, já o DELETE quando deseja-se deletar algum elemento. (ELKSTEIN, 2016)

A segunda coluna do Quadro 3, representa as URIs, que são utilizadas nas chamadas REST, onde antes da barra deve ser adicionado o endereço do serviço, onde o serviço está hospedado. Os elementos que estão entre chave devem ser substituídos por seus respectivos valores. A terceira coluna, descreve o que ocorrerá na chamada, para o respectivo método HTTP e URI.

[9] www.spring.io

Figura 9 - Diagrama de pacotes do WebService



Fonte: Elaborada pelo autor

Quadro 3 - Chamadas REST do serviço de armazenamento de itens do backlog

Método HTTP	URI	Descrição
POST	/backlogs	Salvar um item
GET	/backlogs	Buscar todos os itens
GET	/ backlogs/{id}	Buscar um item por id
PUT	/ backlogs	Atualizar um item
DELETE	/ backlogs/{id}	Deletar um item
GET	/backlog/stream	Notificação de mudança

Fonte: Elaborado pelo autor

Quadro 4- Ferramentas utilizadas no desenvolvimento da aplicação do lado servidor

Nome	Tipo	Versão	Descrição
Eclipse	IDE	Kepler	Ferramenta utilizada na edição e codificação dos serviços
Java	Linguagem de programação e JVM	1.8	Linguagem de programação utilizada
Tomcat	Servidor web Java	7.0.67	Utilizado para executar os serviços no ambiente de desenvolvimento.
Spring	*Framework Java*		Framework utilizado na criação dos serviços

Fonte: Elaborada pelo autor

7 TESTES E ANÁLISE DOS DADOS

Os testes foram utilizados para validar a arquitetura, onde pode ser feita uma análise da sua eficiência nos cenários abordados. A realização dos testes consistiu na elaboração dos casos de teste e criação de um conjunto de testes automatizados para simular o uso da aplicação ScrumTool em três diferentes cenários, que foram: aplicação *online* com banco de dados local, aplicação *off-line* com banco de dados local e aplicação sem banco de dados local.

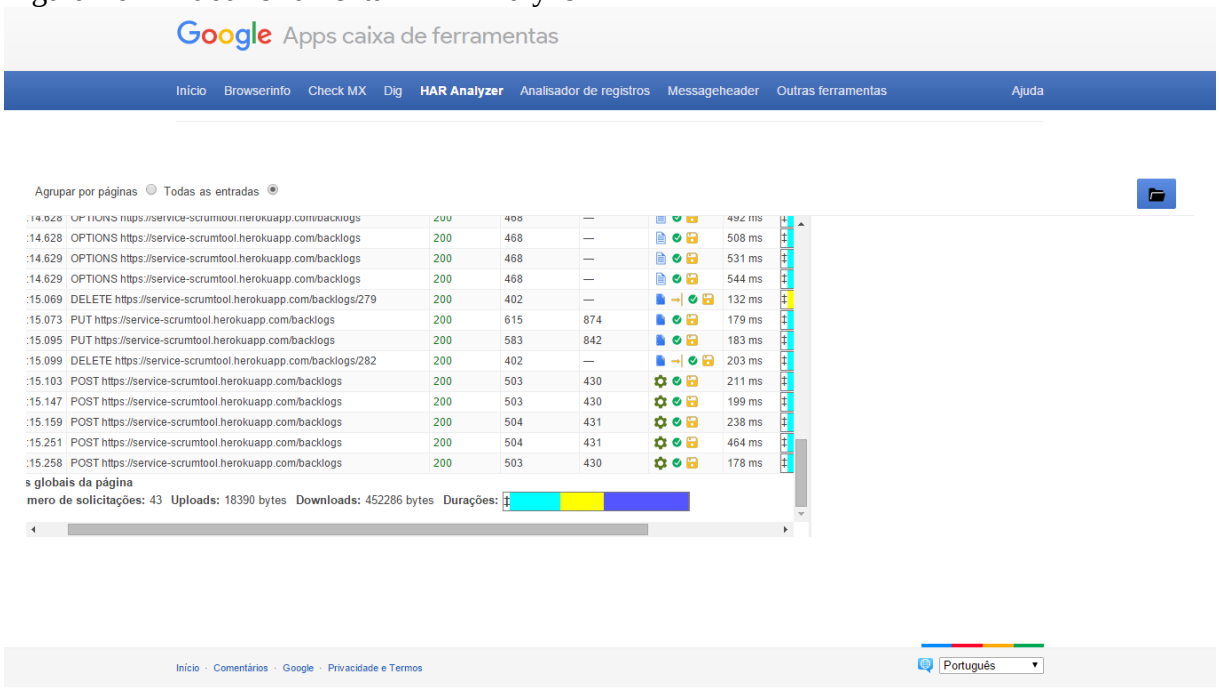
Para a criação do cenário da aplicação off-line com banco de dados local, foi realizada o desligamento manual da rede e posteriormente executado os testes automatizados. Na criação do cenário da aplicação sem banco de dados local, foi criada uma versão da aplicação sem as funções de armazenamento local.

Na elaboração e execução dos casos de teste funcionais automatizados foi utilizada a ferramenta Selenium[10], que é uma ferramenta utilizada na indústria de software para elaboração e execução de testes automatizados. A ferramenta Selenium funciona como um navegador, onde são definidos os cenários e a ferramenta executa os cenários na aplicação e mostra os resultados. Neste trabalho, a ferramenta foi utilizada para criar simulações de uso, que serviram para gerar dados de tráfego na rede.

Nos testes, foram coletados dados relacionados à quantidade de requisições, quantidade de dados que foram feitos *upload* e *download* e tempo total gasto nas requisições. Os dados formam coletados do monitor de rede do Firefox[11]. Os mesmos foram salvos no formato ".har" e posteriormente processados pela ferramenta HAR Analyzer[12]. Um exemplo de análise feita na ferramenta pode ser visto na Figura 10.

[10] seleniumhq.org
[11] http://br.mozdev.org/
[12] https://toolbox.googleapps.com/apps/har_analyzer/

Figura-10 Print da ferramenta HAR Analyzer



Fonte: Elaborada pelo autor

No Quadro 5, podem ser observados os casos de teste que foram elaborados neste trabalho, onde cada caso foi executado nos três cenários apresentados anteriormente. Nos casos de teste, procurou-se testar todas as funcionalidades e algumas simulações de uso.

Quadro 5 - Casos de Teste

Caso de Teste	**Descrição**
CT01	Criação de 5 itens, modificado o status de 2 e deletado 2.
CT02	Carregar a aplicação com 4 itens.
CT03	Carregar a aplicação sem cache dos arquivos HTML, CSS e JavaScript
CT04	Carregar a aplicação utilizando cache dos arquivos HTML, CSS e JavaScript

Fonte: Elaborado pelo autor

7.1 Coleta dos dados

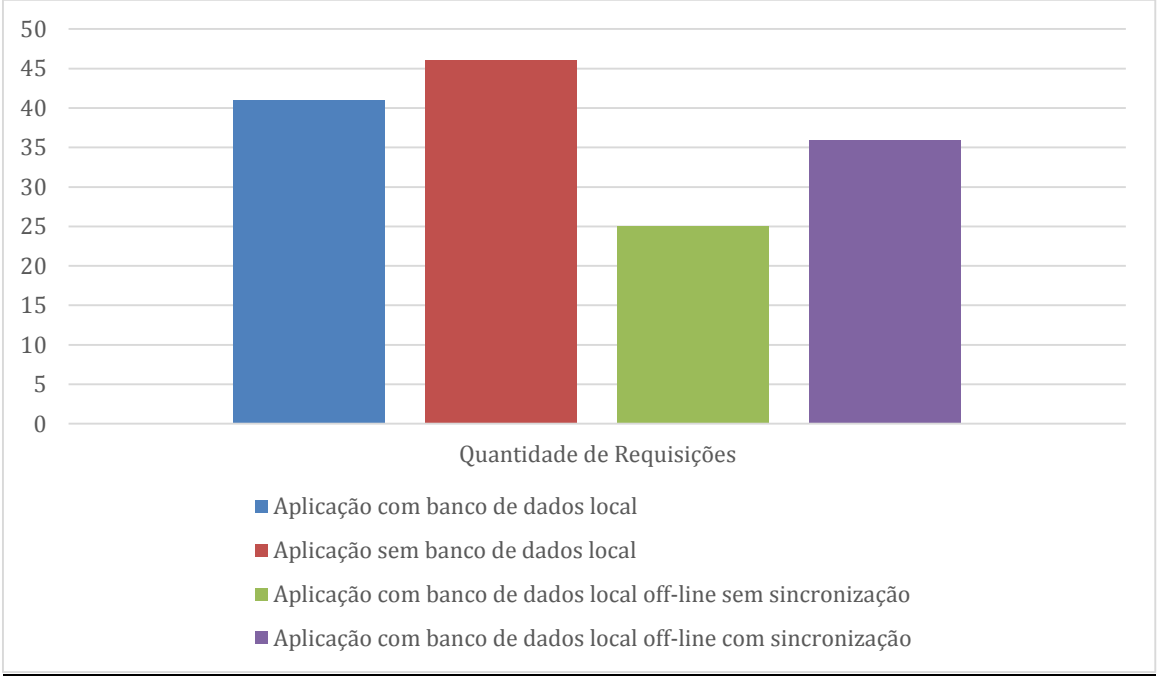
Do Quadro 6 ao Quadro 9, são apresentados os resultados da execução dos casos de teste. No Quadro 10, são a presentados os ambientes de execução das aplicações. Os Gráficos 1,3,5 e 7, apresentam a quantidade de requisição nos cenários CT01,CT02, CT09 e CT04, respectivamente. Os Gráficos 2, 4,6 e 8, apresentam a duração das conexões entre o lado cliente e o lado servidor, nos cenário CT01,CT02, CT09 e CT04, respectivamente.

Quadro 6 - Resultado da execução do CT01

Cenário	Quantidade de Requisições	Upload (bytes)	Download (bytes)	Duração (ms)
Aplicação com banco de dados local	41	18055	452923	8150
Aplicação sem banco de dados local	46	20404	460505	3714
Aplicação com banco de dados local off-line sem sincronização	25	9758	443425	150
Aplicação com banco de dados local off-line com sincronização	36	15044	448978	12386

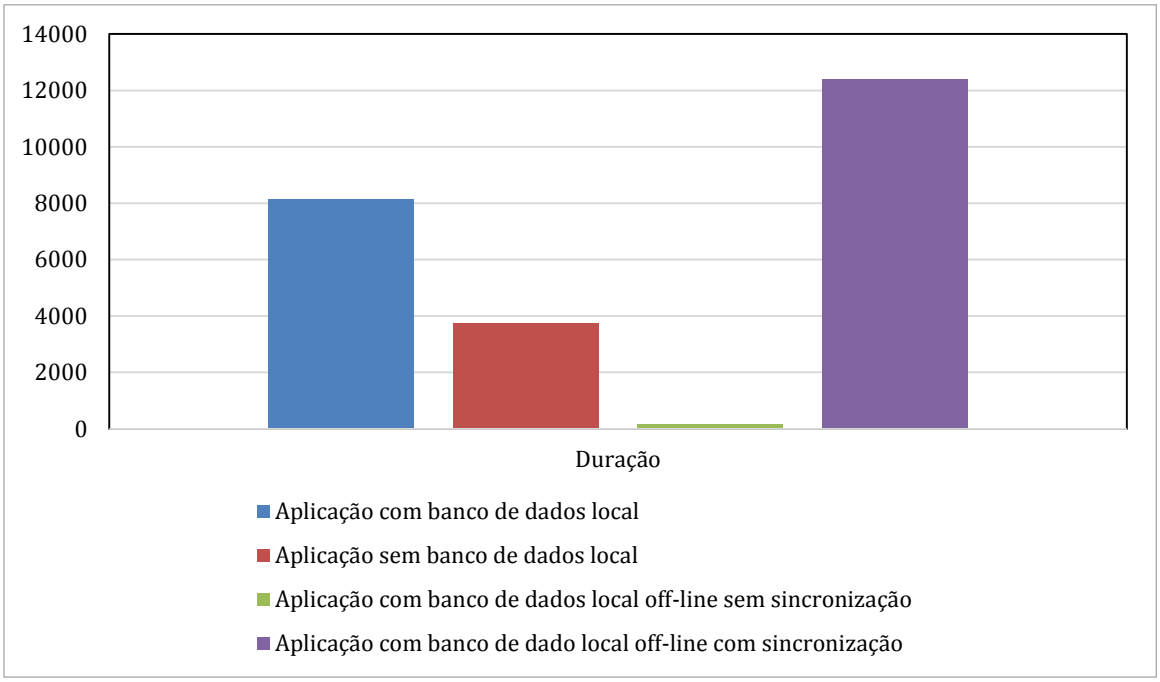
Fonte: Elaborado pelo autor

Gráfico 1 - Quantidade de requisições na execução do cenário CT01



Fonte: Elaborado pelo autor

Gráfico 2 - Duração das requisições na execução do cenário CT01



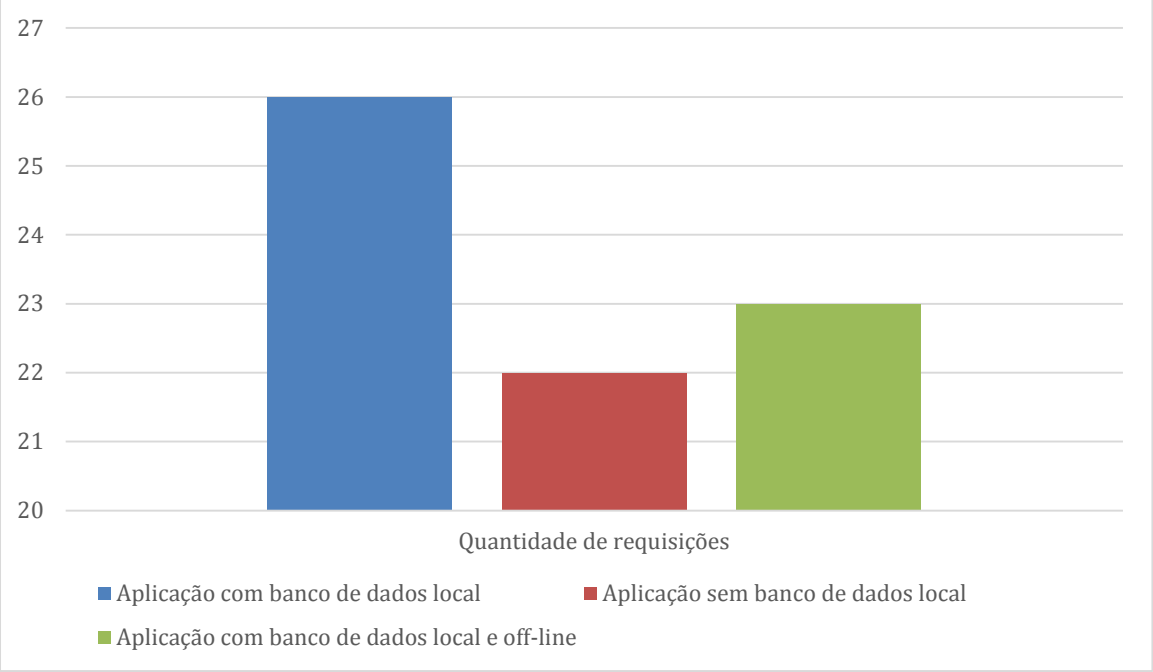
Fonte: Elaborado pelo autor

Quadro 7 - Resultado da execução do CT02

Cenário	**Quantidade de Requisições**	**Upload (bytes)**	**Download (bytes)**	**Duração (ms)**
Aplicação com banco de dados local	26	10431	495657	1707
Aplicação sem banco de dados local	22	8846	446026	603
Aplicação com banco de dados local e off-line	23	8976	442789	2

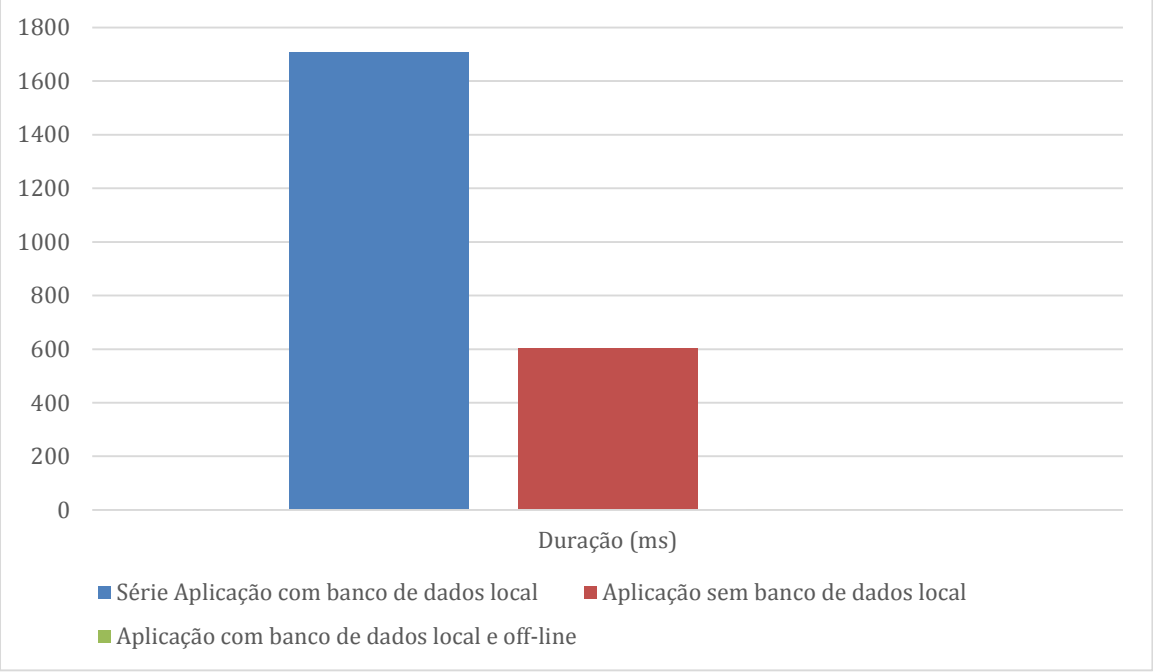
Fonte: Elaborado pelo autor

Gráfico 3 – Quantidade de requisições na execução do cenário CT02



Fonte: Elaborado pelo autor

Gráfico 4 - Duração das requisições na execução do cenário CT02



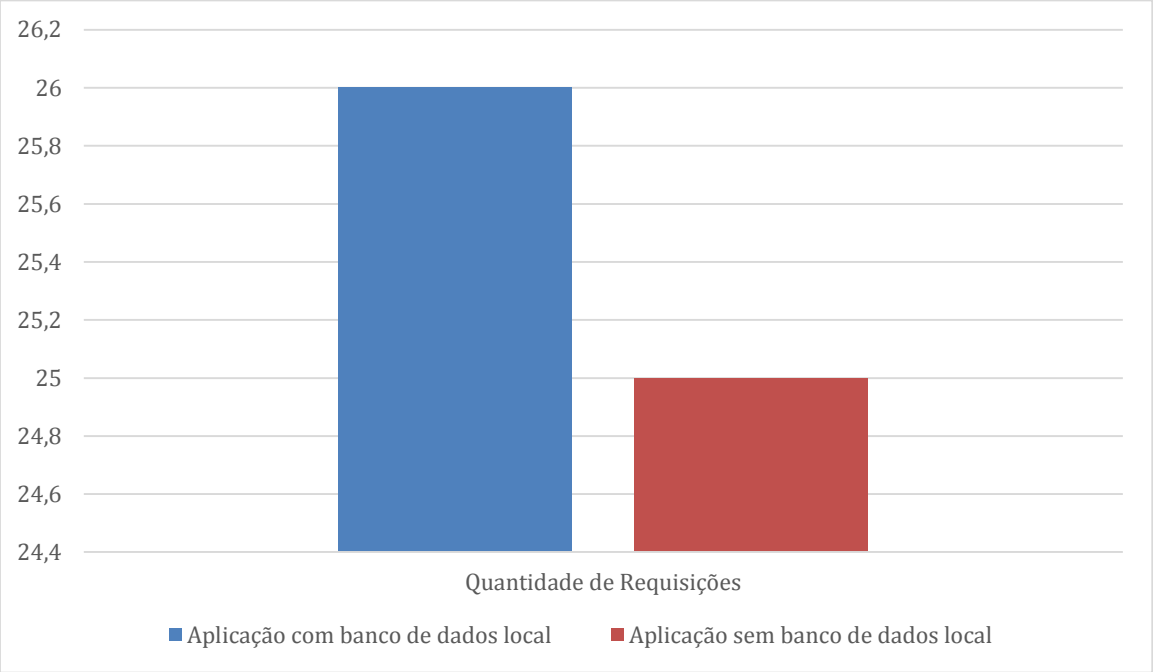
Fonte: Elaborado pelo autor

Quadro 8 - Resultado da execução do CT03

Cenário	Quantidade de Requisições	Upload (bytes)	Download (bytes)	Duração (ms)
Aplicação com banco de dados local	26	10708	180949	48505
Aplicação sem banco de dados local	25	10459	180817	12218

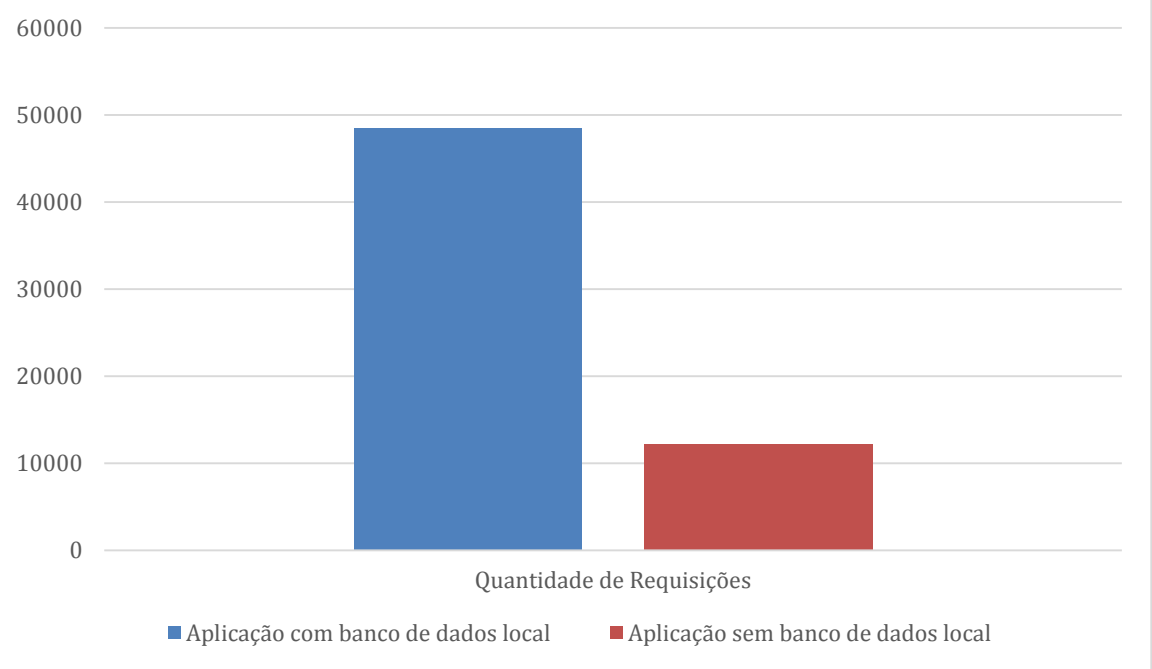
Fonte: Elaborado pelo autor

Gráfico 5 - Quantidade de Requisições na execução do cenário CT03



Fonte: Elaborado pelo autor

Gráfico 6 - Duração das requisições na execução do cenário CT03



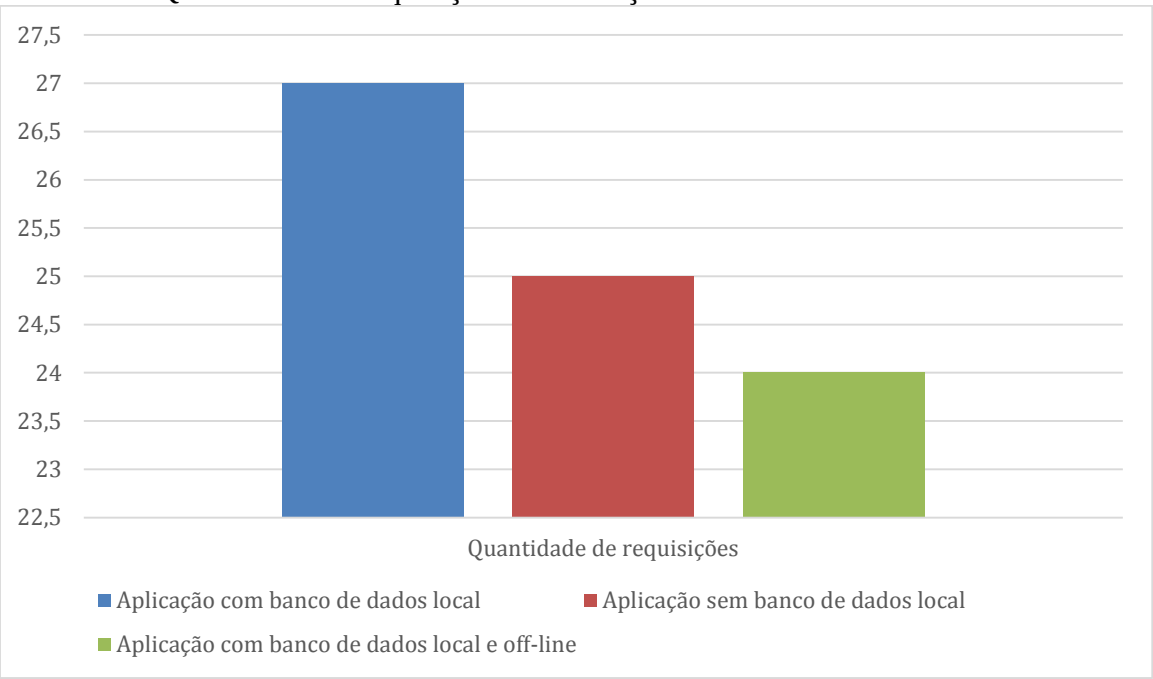
Fonte: Elaborado pelo autor

Quadro 9 - Resultado da execução do CT04

Cenário	**Quantidade de Requisições**	**Upload (bytes)**	**Download (bytes)**	**Duração (ms)**
Aplicação com banco de dados local	27	10860	495714	2463
Aplicação sem banco de dados local	25	10414	495612	2249
Aplicação com banco de dados local off-line	24	9416	442834	7

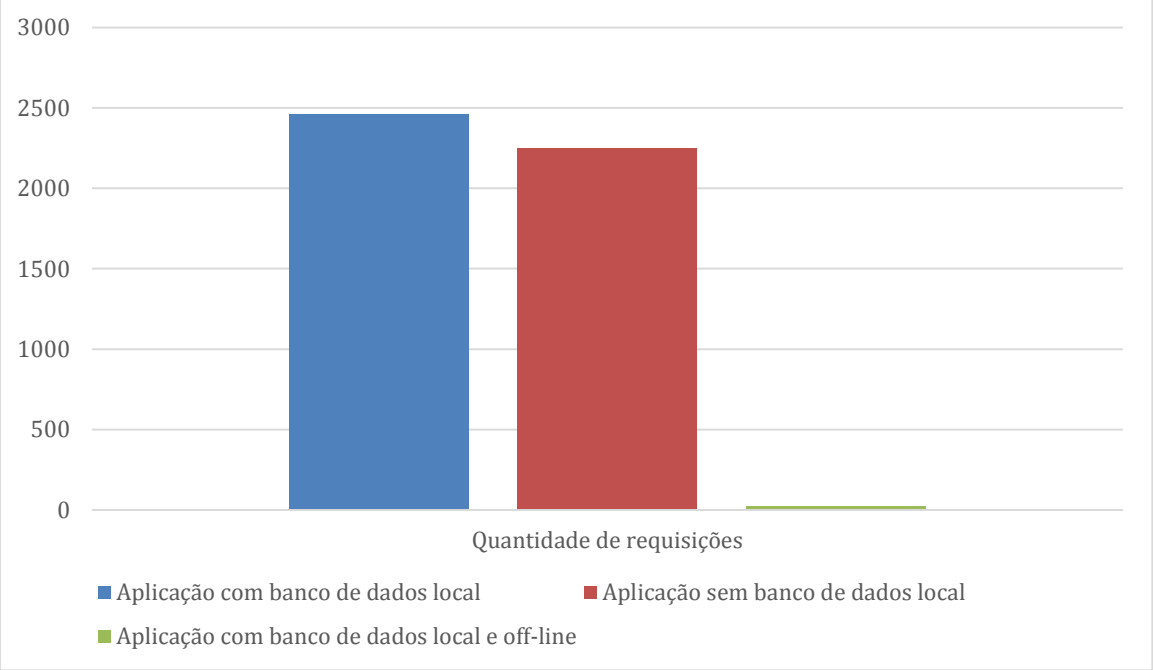
Fonte: Elaborado pelo autor

Gráfico 7 - Quantidade de Requisições na execução do cenário CT04



Fonte: Elaborado pelo autor

Gráfico 8 - Duração das requisições na execução do cenário CT04



Fonte: Elaborado pelo autor

Quadro 10 - Ambiente de execução das aplicações

Aplicação	**Ambiente**	**Configuração**
Aplicação com banco de dados local www.app-scrumtool.rhcloud.com	PaaS openshift	small gear com Tomcat 7
Aplicação sem banco de dados local www.appthin-scrumtool.rhcloud.com	PaaS openshift	small gear com Tomcat 7
Serviço https://service-scrumtool.herokuapp.com	PaaS Heroku	Dynos 512 RAM, Postgres 9.2, Java8

Fonte: Elaborado pelo autor

7.2 Análise do resultado

Com a execução dos testes foi possível verificar que os resultados nos cenários onde a aplicação estava *online* com banco de dados local e aplicação *online* sem banco de dados local, ficaram relativamente semelhantes, não tendo uma grande disparidade entre os mesmos.

No cenário onde não é utilizado banco de dados local a aplicação assemelha-se a uma aplicação cliente servidor tradicional, onde a cada iteração do usuário a aplicação necessita enviar os dados para o servidor.

Quando a aplicação utiliza banco de dados local a quantidade de requisições deveria ser bem menor, devido a uma característica das aplicações RIA. Mas durante análise do resultado dos testes, identificou-se que neste cenário a quantidade de requisições estava sendo maior que nos demais, isso pode ser observado nos Gráficos 3, 5 e 7.

Para identificar o motivo pela qual a quantidade de requisições é maior para este cenário, foi preciso analisar os dados das requisições que haviam sido salvo no formato ".har". Durante a análise das requisições, foi possível constatar que a quantidade de requisições estava maior devido ao mecanismo de sincronização.

O mecanismo de sincronização utilizado para a aplicação de teste consistiu na criação de uma pilha de modificações que era criada cada vez que a aplicação ficava *off-line*, todas as atualizações que ocorriam com a aplicação *off-line* ficavam armazenadas nessa pilha. No momento em que a aplicação ficava *online*, as atualizações eram enviadas para o servidor, uma a uma. Já as alterações que ocorriam no lado servidor eram enviadas para o cliente através

do mecanismo de *Server-Sent Events*. Por conta desse mecanismo, a duração das conexões e a quantidade de requisições aumentaram.

Observou-se então que dependendo da forma como o mecanismo de sincronização tenha sido implementando essa quantidade de requisições tende a diminuir ou aumentar. No caso da aplicação ScrumTool, é possível afirmar que o mecanismo de sincronização não é muito eficiente. Com isso fica claro que o ganho de desempenho proposto pela arquitetura RCWABH5-SOA, depende da forma como o mecanismo de sincronização é implementado.

8 CONSIDERAÇÕES FINAIS

O trabalho proposto teve como objetivo aprimorar a arquitetura RCWABH5, definindo uma arquitetura baseada em SOA para o lado servidor. Como resultado foi definida a arquitetura RCWABH5-SOA e criada uma aplicação afim de validar essa arquitetura.

A arquitetura foi desenvolvida e testada de acordo com as necessidades levantadas no decorrer do trabalho. Para tanto, foi desenvolvida uma aplicação ScrumTool e foram criados cenários de testes. Na etapa de análise do resultado dos testes, percebeu-se que escolhas realizadas no momento de projeto da aplicação, como foi o caso da definição do mecanismo de sincronização, podem afetar o desempenho da aplicação e fazer com que os ganhos propostos pela arquitetura não sejam como esperado.

Com a execução dos testes, identificou-se a deficiência do mecanismo de sincronização utilizado e a dependência que a arquitetura RCWABH5-SOA tem deste mecanismo de sincronização.

Como proposta de trabalhos futuros tem-se a criação de aplicações com arquitetura RCWABH5-SOA e que implementem diferentes mecanismos de sincronização, afim de verificar qual o melhor mecanismo para ser utilizado com essa arquitetura. Outro possível trabalho futuro é a criação de uma aplicação usado a arquitetura RCWABH5-SOA e a criação da mesma aplicação usando outra arquitetura para comparar os resultados das duas.

REFERÊNCIAS

BUSCH, M., KOCH, N. **Rich Internet Applications**. Technical Report 0902, Institute for Informatics, Ludwig-Maximilians-Universität München, Germany.2012.

CHEN, L., LIU, Z. **Design of Rich Client Web Architecture Based on HTML5**, In: International Conference on Computational and Information Sciences, 4. 2012, Chongqing, China.IEEE, p 1009-1012,17-19 aug. 2012.

ELKSTEIN, M. **Learn REST: A Tutorial.** Disponível em http://rest.elkstein.org/. Acesso em: 11 de fev. 2016.

ERL, Thomas. **SOA**: Principios de design de serviço, Editora Pearson Pretice Hall. São Paulo, 2009.

GUO, **Liang at al. Study on GIS architecture based on SOA and RIA**. In: International Conference on Information Sciences and Interaction Sciences (ICIS), 3º edição, 2010. p. 620 – 625.

INFOSYS (India). **Rich Internet Applications**: Opportunities and Challenges for Enterprises. Bangalore, 2007. 7 p.

JOSUTTIS, Nicolai M. **SOA na prática:** a arte da modelagem de sistemas distribuídos, Editora Alta Books, 2008.

MICROSOFT. **Microsoft Application Architecture Guide**. Disponível em https://msdn.microsoft.com/en-us/library/ff650706.aspx. Acesso em: 11 de fev. 2016

PIETRUSZKIEWICZ, W. DZEGA, D. **The Practical Aspects of Rich Internet Application Development and Quality Factors: RIA–based Decision Support System,** West Pomeranian Business School, Faculty of Economics and Information Technology. Szczecin, Poland.2009.

PINA, D.S.A., OLIVEIRA, L. E. M. C. **RIA - Rich Internet Applications: Uma Revisão Dos Principais Expoentes Da Área**, União dos Institutos Brasileiros de Tecnologia. Recife, 2013.

SANTINI, D. **Desenvolvimento De Aplicações Ria Com Javafx ,** In Instituto Federal de Educação, Ciência e Tecnologia Sul-Riograndense - Ifsul, Campus Passo Fundo. 2014.

SOMMERVILLE, Ian. **Engenharia de Software**, 9ª edição. Editora Pearson do Brasil, 2011.

ZHANG, Wenjun. **2-Tier Cloud Architecture with Maximized RIA and SimpleDB via Minimized REST**. In: International Conference on Computer Engineering and Technology (ICCET), 2º edição, 16-18 Abril 2010. p**.** 52 – 56. Chengdu. 2010.

APÊNDICE A - Processo para fazer o *download* do código fonte, instalar e executar a aplicação ScrumTool e o serviço SOA

A aplicação ScrumTool, foi utilizada para validar a arquitetura RCWABH5-SOA, abaixo segue o processo para baixar o código fonte, instalação e execução da aplicação.

Aplicação Cliente:

1- Download e Intalação do git: https://git-scm.com/downloads
2- Abra o terminal do seu computador
3- Faça um clone do projeto:
   git clone https://github.com/OtavioAugustoSousa/ScrumTool.git
4- Abra a pasta “Cliente RIA”
5- Execute o arquivo index.html no seu browser

Serviço:

1- Download e Intalação do git: https://git-scm.com/downloads
2- Abra o terminal do seu computador
3- Faça um clone do projeto que:
   git clone https://github.com/OtavioAugustoSousa/ScrumTool.git
4- Faça o download do eclipse: https://www.eclipse.org/downloads/
5- No Eclipse, escolha a opção importar
6- Selecione a opção: existente maven project
7- Navegue até o local onde você clonou o projeto
8- Abra a pasta “Servico SOA”
9- Clique em “OK”
10- Para executar a aplicação, basta rodar o método main da classe “Application.java”. Não esqueça de configurar o banco de dados, para um banco de dados postgres local(em src/main/resources/application.properties).