CENTRO UNIVERSITÁRIO FACVEST
CURSO DE CIÊNCIA DA COMPUTAÇÃO
TRABALHO DE CONCLUSÃO DE CURSO

APLICAÇÕES PARA A WEB MODERNA COM DART

ÁREA: ENGENHARIA DE SOFTWARE

BRUNO WOLFF

LAGES (SC), NOVEMBRO DE 2012.

Centro Universitário Facvest
Curso de Ciência da Computação
Trabalho de Conclusão de Curso

Aplicações para a Web Moderna com Dart

Área: Engenharia de Software

Bruno Wolff

Projeto apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Ciência da Computação para análise e aprovação.

Lages (SC), Novembro de 2012.

EQUIPE TÉCNICA

Acadêmico

Bruno Wolff

Professor Orientador

Profº. Márcio José Sembay, Msc.

Coordenador de TCC

Profº. Márcio José Sembay, Msc.

Coordenador do Curso

Profº. Márcio José Sembay, Msc.

SUMÁRIO

RESUMO

O objetivo deste trabalho é conhecer o desenvolvimento de aplicações web. Desde sua história, concepção de ideia e o desenvolvimento. A indústria web é a que sofre mais avanços tecnológicos, e que engloba a maior quantidade de usuários de qualquer outra tecnologia atual. A internet, como ferramenta indispensável para o mundo atual, está constantemente em evolução. Sendo assim, a tendência é que cada vez mais, se desenvolvam aplicações web, que permitam uma interação robusta e ágil, pelo usuário. Atualmente existem várias linguagens web, algumas utilizadas há muito tempo, tempo este suficiente para expor algumas de suas falhas de arquitetura, e dar a chance para outras linguagens modernas provarem seu poder. Diante disso, torna-se conveniente o estudo da linguagem Dart, voltada ao desenvolvimento de aplicações estruturadas para a web moderna e um projeto demonstrativo para o mesmo.

Palavras-chave: web moderna, aplicações web, Dart

ABSTRACT

The objective of this paper is to understand the development of web applications. From its history, design and idea. The web industry is suffering more technological advances, and includes the largest number of users than any other current technology. The internet as an indispensable tool for today's world is constantly envolving. Thus, the trend is increasingly to develop web applications, enabling a robust and agile interaction by the user. Currently there are several web languages, some used for a long time, long enough to expose some of its flaws, and give a chance to other modern languages to prove their power. Given this, it is convenient to study the Dart language, built for structured web applications development, and a demonstration project for the same.

Keywords: modern web, web applications, Dart

AGRADECIMENTOS

Aos meus pais, por terem custeado a universidade todos esses anos.
Aos professores, que forneceram conhecimentos variados.
Ao meu colega de trabalho (e líder de equipe) Leonardo Constante Xafranski, que foi paciente e compreensivo comigo, durante a produção deste trabalho.
Aos amigos mais próximos, pela boa companhia dentro e fora do curso.

LISTA DE ABREVIATURAS

AJAX – *Asynchronous JavaScript and XML*

API – *Application Programming Interface*

BSON – *Binary JSON*

CSS – *Cascading Style Sheets*

DHTML – *Dynamic HTML*

DOM – *Document Object Model*

HTML – *Hypertext Markup Language*

HTTP – *Hypertext Transfer Protocol*

JSON – *JavaScript Object Notation*

OS – *Operating System*

PC – *Personal Computer*

PHP – *PHP (Personal Home Page) Hypertext Preprocessor*

RCP – *Rich Client Plataform*

REST – *Representational State Transfer*

RIA – *Rich Internet Application*

ROR – *Ruby On Rails*

RSS – *Rich Site Summary*

VM – *Virtual Machine*

XHR – *XML HTTP Request*

XHTML – *Extensible HTML*

XML – *Extensible Markup Language*

XSLT – *Extensible Stylesheet Language Transformations*

LISTA DE FIGURAS

LISTA DE TABELAS

I INTRODUÇÃO

Construir uma aplicação web pode ser muito divertido, especialmente no começo, quando é provada a experiência de uma gratificação instantânea: codificar, recarregar, repetir.

Infelizmente, finalizando e mantendo uma aplicação *web* não é tão simples. Atualmente, a linguagem dominante no segmento é o Javascript. Ele é incrível para pequenos scripts, e até tem desempenho para rodar aplicações grandes. Mas quando um simples script evolui para uma grande aplicação, o *debugging* e as modificações podem se tornar um pesadelo, especialmente em equipes de desenvolvimento grandes.

Bem-vindo ao Dart, um projeto *open source* que visa permitir os desenvolvedores construírem aplicações mais complexas e de alta performance para a web moderna. Utilizando a linguagem Dart, é possível escrever rapidamente protótipos que podem evoluir de modo significativo, e também conta com o acesso à ferramentas avançadas, bibliotecas ricas, e boas técnicas para engenharia de software.

Logo, temos uma linguagem única, legível, com um estilo familiar a programadores de diferentes cenários. E ainda, características diferenciadas, para uma linguagem designada para adoção em massa.

1.1 Justificativa

Por que o desenvolvimento com Dart?

Javascript tem se mantido ativo por 17 anos. Quando foi introduzido pela primeira vez, não havia *Web 2.0*, Ajax, CSS e, a interatividade *client-side* era pequena. Sua utilização primária era a validação de formulários com caixas de alerta (*alert boxes*).

Com o passar destes 17 anos, a linguagem Javascript evolui de uma linguagem proprietária, mantida e lentamente desenvolvida pela Netscape Communications para um padrão *web* que adiciona novas funcionalidades com regularidade. Porém, tão rápido quanto um grupo pode adicionar novas funcionalidades para um padrão estável, a *Web* evolui infinitamente mais rápida.

E então veio o Dart. A equipe Dart nos questiona: dadas as informações que conhecemos sobre a *Web* hoje, como nós poderíamos construir o Javascript do zero? Como

podemos carregar e rodar o mais rápido possível? Como podemos escrever, de maneira que possamos definir e carregar bibliotecas externas?

Como podemos facilitar para os desenvolvedores escreverem um código “bonito”? Se o Dart é a resposta para todas essas perguntas, então Dart é possivelmente a tecnologia mais excitante a vir ao nosso caminho em um bom tempo.

1.2 Importância

A *Web,* atualmente, já é incrível. É fácil desenvolver pequenas aplicações, que rodem em diferentes plataformas, sem instalações e com uma distribuição quase trivial.

JavaScript é muito flexível e suporta um desenvolvimento incremental. Mas possui algumas surpresas: mesmo com melhorias, ainda existem defeitos de fundamentação na linguagem que impactam na produtividade e alto consumo de tempo para manutenção de *Web Apps* grandes.

Dart tenta melhorar a plataforma web atual, trazendo um melhor suporte para a programação em escala, uma preparação de desenvolvimento mais rápida e, é claro, um poder verdadeiro para manter-se a par do avanço constante de tecnologias.

Deseja-se que com uma linguagem pensada do início, com melhorias substanciais planejadas, seja possível um desenvolvimento mais enxuto para a *Web* Atual.

1.3 Objetivos do Trabalho

1.3.1 Objetivo Geral

Fazer um estudo sobre a *Web 2.0* apresentando os principais pontos de sua evolução, até o desenvolvimento de aplicações web modernas utilizando a plataforma Dart.

1.3.2 Objetivos Específicos

Os objetivos específicos referentes a este trabalho estão descritos a seguir:

- Descrever o atual contexto da *Web*, apontando as principais evoluções e diferenças entre o modelo antigo.

- Descrever a utilização da *Web 2.0*, como plataforma de desenvolvimento para *Web Apps*.

- Entender a aplicação e as formas de desenvolvimento para aplicativos web.

- Apresentar a linguagem Dart, mostrando as principais funcionalidades e características.

- Desenvolver um projeto de demonstração, com finalidade de apresentar as informações e conceitos adquiridos neste trabalho.

1.4 Metodologia

Inicialmente foi realizado um estudo geral sobre a *Web 2.0* (*web moderna*), procurando o entendimento dos avanços ocorridos e possibilidades de utilização.

Num segundo momento, foi feito o estudo de maior profundidade em relação ao desenvolvimento de aplicações web e a plataforma Dart, seus conceitos, características, ferramentas, aplicações, etc.

Após os devidos estudos, foi executada a descrição do referencial teórico, a qual faz parte da pesquisa e suporte ao trabalho apresentado.

Num momento final, foram feitas as descrições do projeto de demonstração, seguidos da execução do mesmo.

1.4.1 Cronograma

O seguinte cronograma foi utilizado para a realização deste trabalho.

Atividades Realizadas	AGO	SET	OUT	NOV	DEZ
Pesquisa					
Confecção e apresentação da pré-proposta					
Elaboração da revisão bibliográfica					
Desenvolvimento do projeto-demonstração					
Entrega do TCC e Artigo					
Defesa do TCC à banca avaliadora					

Tabela 1 *- Cronograma para TCC*

II FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas as principais referências que fazem parte da pesquisa e que dão suporte às soluções apresentadas, estatísticas, história e conceitos sobre as tecnologias que serão utilizadas no decorrer deste trabalho.

2.1 Web 2.0

Segundo O'Reilly (2005), em decorrência das especulações em volta da internet, ocorreu o que foi chamado de "estouro da bolha", uma crise das empresas ".com" nas bolsas de valor, em Abril de 2000. Tal acontecimento marcou uma virada importante na forma como se cria e navega conteúdos online.

Para Alexander (2006), muitas pessoas concluíram que a *Web* estava promovida de maneira exagerada, quando de fato, tal crise mostrava uma possibilidade de ascender e assumir, por parte das empresas de tecnologia, uma posição importante no mercado. Os "pretendentes" corriam atrás, as verdadeiras histórias de sucesso mostravam sua forca, e começou o entendimento do que separa um do outro.

A *Web* se tornou mais robusta, do ponto de vista econômico e por parte da mídia. A quantidade de sites se multiplicou, e as empresas que sofreram com o "estouro da bolha" investiram em novas aplicações e sites mais dinâmicos, como recurso para aumentar o numero de usuários.

Esta tendência foi denominada por O'Reilly (2005) e pela MediaLive International como Web 2.0.

> A mudança para uma internet como plataforma, e um entendimento das regras para obter sucesso nesta nova plataforma. Entre outras, a regra mais importante é desenvolver aplicativos que aproveitem os efeitos da rede para se tornarem melhores quanto mais utilizados pelos usuários, aproveitando a inteligência coletiva. (O'REILLY, Tim. **What Is Web 2.0?** 2005).

2.1.1 A Web como Plataforma

Como muitos conceitos importantes, a *web* não tem limites estabelecidos. Para O'Reilly (2005), é possível visualizar a *Web 2.0,* como um conjunto de princípios e práticas, comuns a inúmeros sites. Tais características nos ajudam a entender e classificar um serviço como pertencente à *Web 2.0*.

Temos então, uma viabilização de funções *online*, que antes eram somente possíveis com a utilização de aplicativos *desktop*.

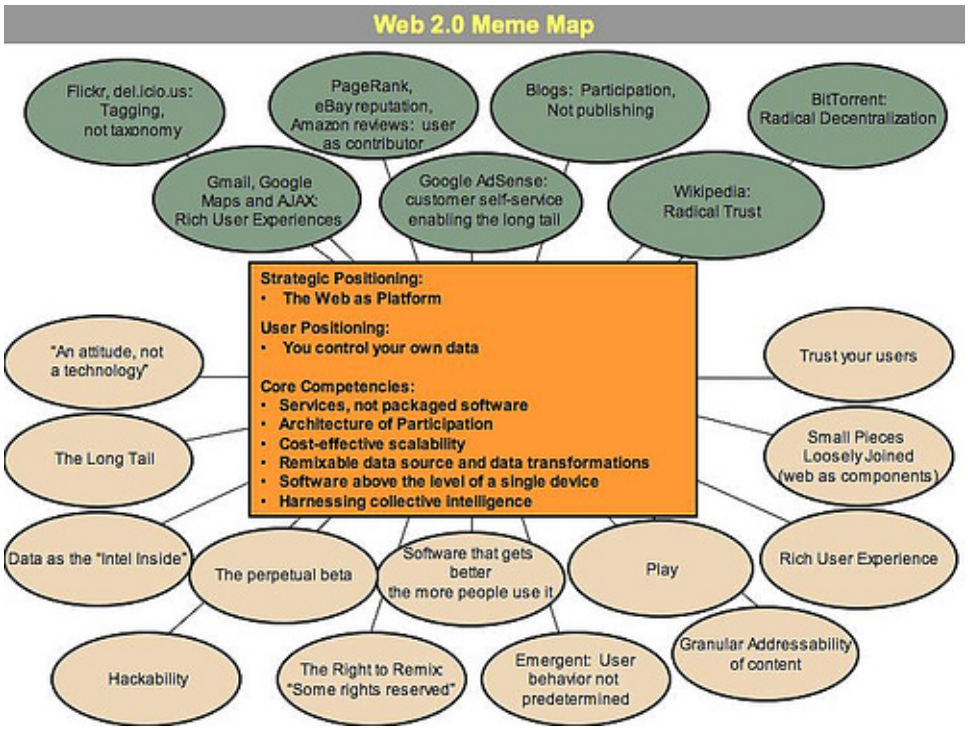

Figura 1 - *Mapa da Web 2.0, com seus conceitos e noções (O'Reilly , 2005)*

Existem versões de aplicativos *desktop* que agora são um serviço *online*, sem necessidades de instalação e/ou requisitos extras, bastando apenas um navegador moderno e compatível. O usuário tem a possibilidade de realizar tarefas diversificadas, podendo armazenar seus dados na "nuvem", e sem questões de dependência para com ambientes de trabalho e afins.

2.1.1.1 Netscape vs. Google

Segundo O'Reilly (2005), se o Netscape era o padrão para a *Web 1.0*, então o Google é certamente o padrão para a *Web 2.0.*

O Netscape enquadrou "a *web* como plataforma" nos termos de paradigma de *software*: o principal produto deles era um *browser*, uma aplicação *desktop*, e a estratégia deles era usar sua dominância na "competição" entre navegadores, para estabelecer um mercado com produtos para servidores de alto preço. Na teoria, o controle dos padrões para apresentação de conteúdo e *web applications*, daria ao Netscape o tipo de poder apreciado pela Microsoft no mercado de PCs.

O Netscape promoveu uma *webtop*, para substituir o *desktop*, e planejou sua população de informações e *applets* com dados enviados diretamente pelos *information providers* que então iriam adquirir servidores do Netscape.

Para O'Reilly (2005), no final, tanto *web browsers* quanto *web servers* se tornaram comodidades, e o valor em si das informações transportado acima deles, entregue através da plataforma web.

O Google, em contraste, já começou como uma plataforma *web* nativa, nunca vendida ou "empacotada", mas entregue como um serviço, com clientes pagando, diretamente ou indiretamente, para o uso do mesmo. Nenhumas das armadilhas da indústria de software antigo estão presentes. Sem liberações agendadas, somente melhoramento contínuo. Sem licenciamento ou venda, somente utilização. Sem *porting* para plataformas diferentes, para os usuários rodarem o programa no próprio equipamento, somente uma massiva coleção escalável de computadores rodando OSs *open source*, juntamente com aplicações incubadas e utilitários, que ninguém fora da companhia sequer verá.

Porém, o Google requer uma competência tecnológica que o Netscape jamais precisou: gerenciamento de *database*. De acordo com O'Reilly (2005), o Google não é somente uma coleção de ferramentas, é um *database* especializado. Sem os dados, as ferramentas são inúteis; sem o *software*, os dados não-gerenciáveis.

O serviço do Google ocorre entre o *browser* e o motor de buscas (e servidores de destino de conteúdo), como um mediador entre usuário e suas experiências online.

Enquanto ambos Netscape e Google poderiam ser descritos como *software companies*, é claro que o Netscape pertenceu ao mesmo mundo de *software* que a Microsoft, Oracle, Apple. Enquanto os "companheiros" do Google são outras aplicações de internet,

como Amazon, eBay, etc.

2.1.1.2 Uma Plataforma sempre vence uma Aplicação

Antigamente, não era raro o confronto de plataformas com aplicações (por exemplo, Microsoft Office vs. Lotus 1-2-3, WordPerfect). Porém, para O'Reilly (2005), desta vez a "briga" não é entre uma plataforma e uma aplicação, mas entre duas plataformas, cada uma com modelo de negócio totalmente diferente.

De um lado, o modelo padrão *Desktop*, com uma enorme base de adoção, altamente integrado com o sistema operacional e com controle de APIs, o qual tem o controle sobre o paradigma da programação. Do outro lado, um sistema sem dono, integrado através de uma série de protocolos, *open standards* e acordos para cooperação.

O Windows é um brilhante exemplo de sucesso para a solução de problemas no começo da "era PC". Ele nivelou sua estrutura para desenvolvedores de aplicações, eliminando situações que antes atormentava a indústria.

Mas, tal abordagem monolítica, controlada por somente um fornecedor não é mais uma solução, é um problema. Sistemas orientados a comunicação, como a web (plataforma) certamente são a chave.

As companhias que tiverem sucesso na Web 2.0, serão as que entenderão as regras do novo jogo.

2.1.2 Aproveitando a Inteligência Coletiva

Segundo O'Reilly (2005), o principal motivo por trás do sucesso das grandes empresas da era Web 1.0 que sobreviveram para liderar a Web 2.0, parece ser o fato das mesmas terem adotado o poder da *web* para aproveitar a inteligência coletiva.

O *hyperlinking* é a fundação da web. Enquanto usuários adicionam novos conteúdos, e novos sites, tudo isso é conectado na estrutura da *web* através de outros usuários descobrindo e "linkando".

O próprio motor de busca do Google, que rapidamente se tornou um líder no segmento, baseou-se em métodos para avaliação da utilização da estrutura de *links* na *web*,

para trazer melhores resultados aos usuários.

O principal produto do eBay é a atividade coletiva de seus usuários; como a própria web, o eBay cresce em resposta às atividade dos usuários. Com uma enorme quantidade de vendedores e compradores, tal vantagem de mercado torna a entrada de serviços semelhantes no segmento, no mínimo pouco atrativa.

Companhias mais recentes e inovadoras que conseguiram captar tal detalhe estão deixando sua marca na web. Como exemplo, temos a Wikipedia, uma enciclopédia *online* baseada na improvável ideia de que um conjunto de dados pode ser adicionado por qualquer usuário da *web*, e editado por outro qualquer. É uma experiência de confiança, onde ocorre uma dependência de todos para a finalização de uma informação correta.

Os efeitos das contribuições de usuários são a chave para a dominância de mercado na Web 2.0. Saber utilizar o coletivo, para o coletivo.

2.1.2.1 Blogging e a Sabedoria das Multidões

Uma das principais características marcantes da Web 2.0, é a ascensão dos *blogs*. Claro que *home pages*, e diários pessoais estão por ai há muitos anos, desde os tempos antigos da web, então qual seria a novidade sobre isso?

Na sua forma mais básica, um *blog* é apenas uma página pessoal, em formato de diário. Porém, como Skrenta (2005) destaca, *blogs* podem até parecer como páginas HTML comuns, mas a principal diferença é que eles são organizados cronologicamente.

Parece uma diferença trivial, mas isso leva a uma cadeia totalmente diferente de publicidade e entrega de valor (conteúdo).

Uma das coisas que têm feito a diferença (principalmente nos *blogs*) é uma tecnologia chamada RSS. A possibilidade de alguém se inscrever em uma página, ter notificações para cada nova atualização, procurar conteúdos organizados de forma cronológica.

Segundo Skrenta (2005), tal conceito é chamado de “web incremental”, ou também “web viva”.

Se uma parte essencial da Web 2.0 é o aproveitamento da inteligência coletiva, tornando a web em uma espécie de cérebro global, a “blogosfera” é o equivalente a uma constante vibração mental, é aquela voz que nós escutamos dentro da cabeça. E como reflexo

de um pensamento consciente, a “blogosfera” começou a ter um efeito poderoso.

Primeiro, porque os motores de busca baseiam-se em questões de relevância (usam estruturas de links para “adivinhar” seus resultados), blogueiros, como os mais prolíficos e “linkadores oportunos”, tem um papel desproporcional na formação de resultados para pesquisas. Segundo, porque a comunidade de blogs é tão auto-referencial, que blogueiros prestando atenção em outros, magnificam sua visibilidade e poder. A “câmara de eco” que os críticos condenam, também é um amplificador.

Para O'Reilly (2005), se os blogs fossem meramente “amplificadores”, não seriam interessantes. Mas assim como a Wikipedia, os blogs aproveitam a inteligência coletiva como uma espécie de filtro, é a sabedoria das multidões.

2.1.3 Fontes de Dados Ricas e Especializadas

Toda aplicação web significante atualmente tem sido apoiada por um *database* especializado. O gerenciamento de dados é um ponto essencial de competência para as companhias da Web 2.0, tanto que em alguns momentos é utilizado o termo “*infoware*” no lugar de *software*.

Numa era digital, quem possui o controle de certos dados, acaba consequentemente controlando um mercado também. Para Brodkin (2005), embora tenha sido argumentado que a vantagem de negócios via controle de APIs de software é muito mais difícil na era da internet, o controle das fontes de dados não, especialmente se essas fontes são caras para criar ou passíveis de retornos crescentes através dos efeitos da rede.

Existe uma corrida para dominar alguns tipos de informação, e em vários casos onde é significativamente caro de se criar a fonte de dados, talvez exista a possibilidade de adoção de um modelo dominante e de exclusividade para tais dados.

Segundo O'Reilly (2005), os vencedores são aqueles que atingem um nível crítico de massa através da agregação de usuários e suas interações, e saiba tornar os dados agregados em um serviço de sistema.

2.1.4 Fim dos ciclos para lançamento de Software

Uma das características definidoras do *software* da era Web 2.0, é ele ser entregue como um serviço, não como produto. Para O'Reilly (2005), tal fato leva à um número de mudanças fundamentais no modelo de negócio.

A primeira grande mudança é a expertise em realizar operações diárias. Tão fundamento é a mudança de "software como artefato" para "software como serviço", que o software deixará de executar a menos que seja mantido numa base diária. Por exemplo, o motor de buscas Google, deve continuamente rastrear a *web* e atualizar seus índices, filtrar spams e outras tentativas de influenciar seus resultados, de forma contínua e dinamicamente responder centenas de milhões de consultas de usuários assíncronos, simultaneamente, combinando-os com o contexto adequado à publicidade.

Talvez seja por isso que as técnicas de administração de redes, sistemas e balanceamento de cargas do Google sejam mais bem guardadas que os próprios algoritmos de busca. O sucesso na automatização desse processo é a chave de sua vantagem sobre os competidores.

Também não é coincidência o fato de linguagens de script (como Perl, Python, Ruby, e agora Dart) terem um papel importante nas companhias da era Web 2.0. Essas linguagens dinâmicas, antes vistas de forma crítica, são as escolhas para os administradores de sistemas e redes, assim como desenvolvedores de aplicações construindo sistemas dinâmicos que requerem mudanças constantes.

A segunda grande mudança, segundo O'Reilly (2005), é o tratamento do usuários como co-desenvolvedores, em reflexo das práticas de desenvolvimento open source (mesmo que o *software* em questão não seja lançado sobre uma licença open source).

O "beta perpétuo", um desenvolvimento contínuo, apoiando-se na intervenção dos usuários para detectar as possíveis falhas. Neste padrão, a programação se torna uma atividade contínua, o software é corrigido sem necessidade de reinstalações. O termo "beta perpétuo", para O'Reilly (2005), indica que o *software* está em constante fase de testes e constante evolução. Isso demonstra que o software deixou de ser um artefato para se tornar um processo de comprometimento com os usuários, que apontam falhas, as quais devem ser corrigidas.

Companhias nativas da Web 2.0, aproveitam uma vantagem natural sobre outras que ainda possuem padrões antigos (na sua fundação), e modelos de negócio correspondentes.

2.1.5 Modelos de Programação Leve

Uma vez que a ideia de serviços web se tornou aplicável, grandes empresas adotaram a "moda", com uma complexa pilha de serviços web projetada para criar ambientes de programação altamente confiáveis para aplicações distribuídas.

Segundo O'Reilly (2005), um ponto importante do sucesso dessas empresas, foi o apoio ao modelos leves de programação, que permitem o desenvolvimento de sistemas fracamente acoplados. A complexidade da pilha de *web services* patrocinados pelas companhias é projetada para ser altamente acoplada. Enquanto esse modelo é necessário em vários casos, várias das aplicações mais interessantes podem realmente permanecer flexíveis e até mesmo "frágeis". Lembrando, que mentalidade da Web 2.0, é muito diferente do padrão convencional de TI.

Skrenta (2005) observa que web *services* simples (como RSS e os baseados em REST) é sobre disponibilizar dados para fora, não controlar o que acontece do outro lado. Esta é uma ideia fundamental, um reflexo do que é conhecido como "princípio fim-a-fim".

Os modelos de negócio leves são companheiros naturais das linguagens de programação e conexões "leves". A Web 2.0 é boa na ideia de reutilização, desenhada para o compartilhamento e modificação livre, permitindo o usuário copiar códigos facilmente, e continuar com os avanços.

O'Reilly (2005) destaca, que graças ao "conceito leve" é fácil construir um serviço mesmo sem ter um modelo de negócio. A comodidade de componentes é abundante, e criar valor (conteúdo) sabendo se utilizar de meios já existentes nos leva a um ponto importante da Web 2.0: "*innovation in assembly*".

2.1.6 Software acima do nível de "um Único Dispositivo"

Uma outra característica importante da Web 2.0, é o fato de não haver mais uma limitação exclusiva à plataforma PC. Para O'Reilly (2005), softwares úteis escritos acima do nível de "um único dispositivo", vão liderar altas margens por muito tempo.

Claro que as aplicações web se encaixam nesse contexto, pois até mesmo a mais simples aplicação web envolve no mínimo dois computadores: uma fazendo o *hosting* do *web server* e outro, o *hosting* do *browser*. Porém a área de desenvolvimento se estende para outras

maiores e mais amplas: integrações com dispositivos móveis, embarcados, etc.

Um exemplo real seria o Microsoft SkyDrive, serviço de armazenamento de dados na nuvem. Existe desde uma aplicação cobrindo dispositivos de mão, até aplicativos *desktop* multiplataforma, esses servindo apenas como estações de controle de um serviço invisível ao usuário.

Segundo Brodkin (2007), essas são uma das áreas da Web 2.0 onde existe um aguardo por grandes mudanças, mais inovadoras para mais plataformas/dispositivos. As capacidades de desenvolvimento para o futuro são ilimitadas.

Antigamente ocorria um desenvolvimento com garantias para somente uma plataforma, enquanto na Web 2.0 ocorre para várias, podendo ocorrer fracassos para algumas, por questões de tecnologia específica e até mesmo publicidade fraca, e sucesso para outras.

2.1.7 Experiências Ricas para o usuário

Antigamente a Web era usada para entregar *applets* e outros tipos de conteúdo para dentro do navegador, basicamente por meio de páginas HTML, no modelo "clique, aguarde e recarregue", ou seja, a página é enviada do servidor para o cliente e recarregada a cada evento, envio de dados ou navegação. Este modelo de comunicação é conhecido como síncrono, e representado pela **Figura 2**.

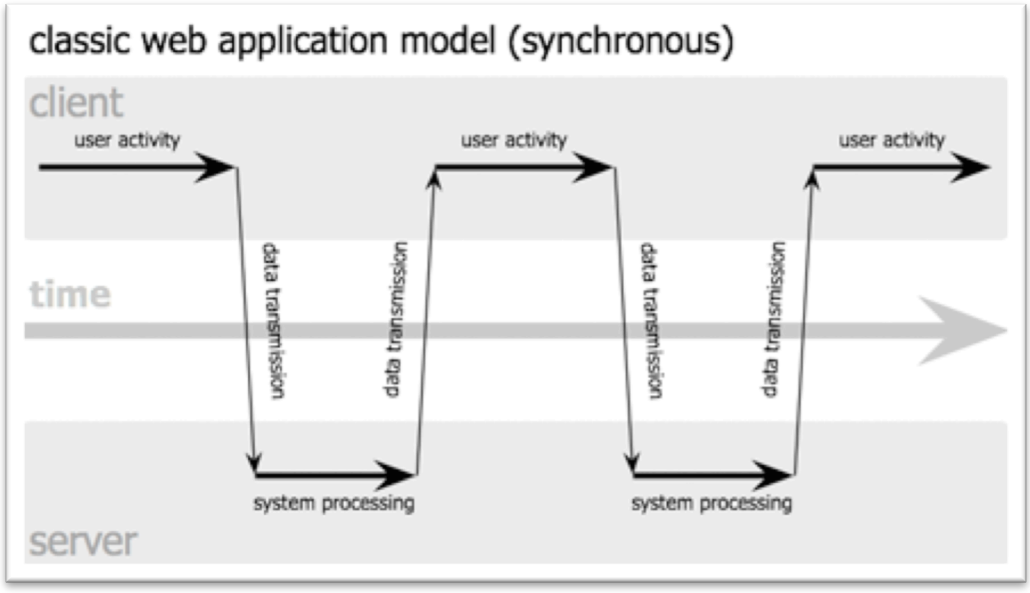

Figura 2 - *Modelo clássico de Aplicação Web (síncrona) (Garret, 2005)*

A introdução do Java foi marcada em volta de tais conteúdos. Logo veio o JavaScript e então o DHTML, que introduziram um modelo leve de prover programação *client-side*, trazendo experiências ricas para o usuário.

Surge então o termo RIA (evangelizado pela Macromedia para ressaltar a capacidade do Flash de entregar não só conteúdo multimídia, mas também aplicações com interface), que define *web apps* como possuidoras de característica e funcionalidades de software tradicionais do tipo *desktop*, transferindo todo o processamento da interface para o navegador (cliente) e se comunicando com o conteúdo de dados (servidor).

No RIA o modelo de comunicação é assíncrono, permitindo um fluxo de dados na rede significativamente reduzido, levando os usuários a ter a sensação de estarem em uma aplicação desktop.

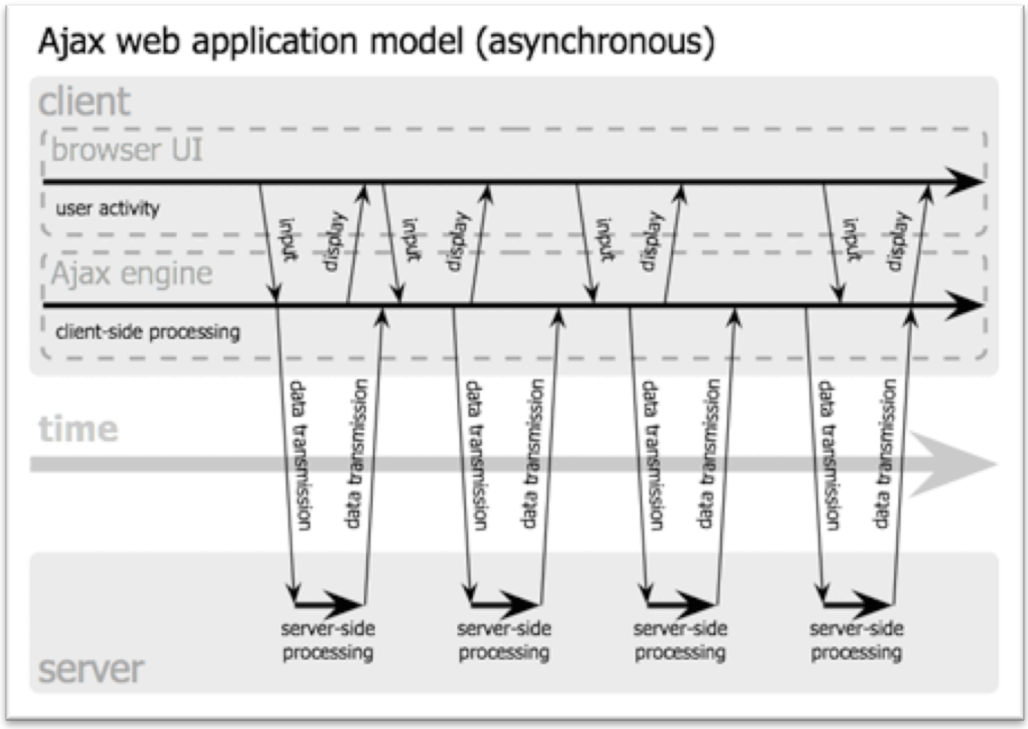

Figura 3 - *Modelo Ajax de Aplicação Web (assíncrona) (Garret, 2005)*

Porém, o potencial da web para entregar aplicações de grande escala não atingiu a grande massa até a introdução do Gmail, seguido pelo Google Maps, ambos web apps, com interfaces ricas e alta interatividade.

O Google utilizou o Ajax no desenvolvimento, e como descrito por Garret (2005), Ajax não é uma tecnologia. É realmente várias tecnologias, cada uma trabalhando da sua maneira, juntas de uma forma poderosa.

O Ajax incorpora:

- padrões de apresentação, utilizando XHTML e CSS;
- visualização dinâmica e interação usando o DOM;
- manipulação e troca de dados usando XML e XSLT;
- recuperação de dados assincronamente por XHR;
- e JavaScript unindo tudo.

O Ajax também foi (e ainda é) um componente chave de várias outras aplicações Web 2.0. Os desenvolvedores web finalmente podiam criar web apps tão ricas quanto as versões *desktop*.

Graças à estas tecnologias e avanços, foi possível construir uma web moderna, preparada para atualizações e melhorias.

2.2 Desenvolvimento de Aplicações Web (Web Apps)

As pessoas olham para a Internet, como uma fonte de obtenção produtiva e rentável de informações. Para Zalewski (2012), desde o início, o objetivo tecnológico da internet tem sido uma forma de utilizar ela como um meio universal para troca de informações através das redes de computadores.

Uma aplicação *web* é um aplicativo que é rodado por meio do navegador, através de uma conexão com a Internet ou uma intranet. Um exemplo para ilustrar tal conceito, seria uma loja *online* acessada via o navegador Google Chrome, como a Amazon.com, que desenvolveu uma aplicação "*web store*" proprietária para vender livros e outros produtos *online*.

2.2.1 Como projetar uma Aplicação Web

O desenvolvimento de uma aplicação web, contém 4 fases principais:

- Prevendo a natureza e direção do projeto

- Elaboração do plano
- Desenvolvimento
- Testando, suporte e estabilidade

2.2.1.1 Prevendo a natureza e direção do projeto

Maccaw (2011) define, que em um modelo atual, a gestão e os colaboradores designados para o projeto devem estabelecer metas que o projeto deve alcançar. Uma abordagem comum para projetos de aplicações web é ignorar as demandas de públicos específicos e fazer o trabalho para atender a todos, concentrando-se no fluxo de atividade, e não no usuário específico.

Segundo Zalewski (2012), o truque para projetar um suporte eficaz para uma atividade, é projetar uma visão rica sobre o que é a atividade, o que ela envolve e o que os participantes estão tentando realizar.

2.2.1.2 Elaboração do plano

Uma equipe designada determina as funcionalidades do aplicativo, quais recursos devem ser incluídos, qual linguagem utilizar, e o tempo de duração do projeto. É importante entender o que os usuários desejam, mas a adesão a todos os seus pedidos pode levar a projetos excessivamente complexos, e que se torna pior a cada revisão.

Toda atividade humana é hierárquica e pode ser entendida em três níveis de análise: atividade, ação e operação. É necessário um mapa de atividade, para representar todos os encaixes do projeto, e fornecer um retrato rico de sua composição em termos de ações e operações. Assim o aplicativo é projeto de acordo.

2.2.1.3 Desenvolvimento

No desenvolvimento de uma aplicação, é onde os programadores web começam a codificação, testes e publicação dos dados. Isto é, quando coisas específicas de tecnologia são

definidas. Variáveis de dados, entidades e procedimentos são estabelecidos.

Segundo Strom (2012), as aplicações *web* são geralmente divididas em blocos lógicos chamados de "*tiers*" (níveis), onde cada camada tem seu papel. As aplicações tradicionais consistem de apenas um nível, que reside na máquina do cliente, mas os aplicativos web se prestam a uma abordagem em camadas por natureza. Embora muitas variações sejam possíveis, a estrutura mais comum é a aplicação de três camadas.

As três camas são chamadas de aplicação, apresentação e armazenagem, por essa ordem. Um navegador é o primeiro nível (apresentação), um motor com alguma tecnologia de conteúdo dinâmico da Web (como ASP.NET, ColdFusion, PHP, ROR ou Dart) é a camada intermediária, e um banco de dados é o terceiro nível (armazenamento). O navegador envia pedidos para a cama intermediária, que atende fazendo consultas e atualizações no banco de dados, retornando uma interface rica para o usuário.

Zalewski (2012) lembra, que para aplicações mais complexas, uma solução de três camadas, por se sair curta, talvez seja necessários ajustes na abordagem de camadas, quebrando a lógica de negócios residente na camada de aplicação.

2.2.1.4 Testando, suporte e estabilidade

Como aplicações web são, na sua grande maioria, totalmente dependentes de uma conexão, as mesmas devem ser exaustivamente testadas utilizando ferramentas profissionais, estabelecendo políticas e procedimentos para um suporte bem sucedido.

Aplicações web sofrem a mesma série de testes de integração e unidade que os modelos tradicionais desktop. Mas por terem uma variação maior de ambiente, alguns testes adicionais são realizados, tais como:

- segurança
- desempenho, carga, stress
- validação de HTML / CSS
- acessibilidade
- usabilidade
- funcionalidade em navegadores diferentes

2.2.2 Vantagens e Desvantagens do desenvolvimento de Aplicações Web

Fowler e Stanwick (2004) destacam como benefícios do desenvolvimento web, o fato de "*web applications*" geralmente terem poucos requerimentos no lado cliente, onde upgrades são feitos automaticamente, e se integram com outros serviços de forma fácil, sem interação do usuário para configurações complexas. A compatibilidade entre diferentes navegadores também é um grande benefício.

Porém, algumas desvantagens ainda são evidentes, como o fato dos aplicativos estarem fisicamente em servidores remotos. Portanto, quando a conexão é interrompida, a aplicação não é mais utilizável (atualmente existem aplicativos que consegue manter um nível de funcionalidade mesmo sem internet, mas ainda, sempre dentro do navegador).

Para Garret (2005), assim como aplicações *desktop* tradicionais, as verões web possuem diferentes níveis de risco. Uma simples página pessoal é muito menos problemática do que, por exemplo, um sistema de negociações para a bolsa de valores. Para alguns, a segurança e possíveis falhas de software, são grandes problemas. Por isso, um tempo dedicado para documentações, planejamento de testes, controle de mudanças, análises de requerimentos e descrições de arquitetura são práticas que ajudam na redução de riscos.

2.3 Dart

Dart é uma linguagem de programação estruturada, *open source e* puramente orientada à objetos.

Desenvolvida pelo Google para a criação de aplicações web complexas baseadas em navegadores *web* (*browsers*). É possível rodar aplicações criadas em Dart tanto utilizando um navegador que suporte o código Dart quanto compilando para JavaScript.

A linguagem possui uma sintaxe familiar, é baseada em classes, com especificação de tipos opcional e *single threaded*. Contém um modelo de simultaneidade chamado *isolates* que permite execuções paralelas.

Além de rodar em navegadores web e poder ser convertida para JavaScript, é possível também rodar o código Dart em linha de comando, hospedada na Dart VM, permitindo tanto as partes *client* e *server* do aplicativo serem codificadas na mesma linguagem.

A sintaxe da linguagem em si é muito semelhante ao Java, C#, e JavaScript. Segundo Buckett (2012), uma das metas para o desenvolvimento da linguagem, era que ela parecesse familiar.

Abaixo um pequeno script em Dart, compreendendo de apenas uma única função chamada *main*.

```
main() // # A
{
  var d = "Dart"; // #B
  String w = "World"; // #C
  print("Hello ${d} ${w}"); // #D
}

/*

A -> Ponto único de entrada da função, é executado
quando o script é totalmente carregado.

B -> Tipagem opcional (sem especificação de tipos)

C -> Anotação de tipo (com especificação de tipo, String)

D -> Utiliza a interpolação de strings para mostrar o
texto "Hello Dart World" no console

*/
```

Figura 4 - *Exemplo de um "Hello World" em Dart*

Este script (**Figura 4**) pode ser embutido dentro de uma página HTML, e rodar dentro do navegador Dartium (uma versão para desenvolvedores Dart do navegador Google Chrome). É possível converte-lo para JavaScript através do utilitário dart2js, garantindo a compatibilidade entre os navegadores modernos, ou rodar o script diretamente pela linha de comando (*server-side*), através da máquina virtual própria (Dart VM) da linguagem.

Porém, para Buckett (2012), existe mais do que apenas a linguagem. A figura abaixo mostra o ecossistema de ferramentas, o qual inclui múltiplos "*runtime environments*", linguagem, ferramentas de edição e um boa quantidade de bibliotecas, criadas para otimizar o

ciclo de trabalho para desenvolvimento de *Web Apps* complexos.

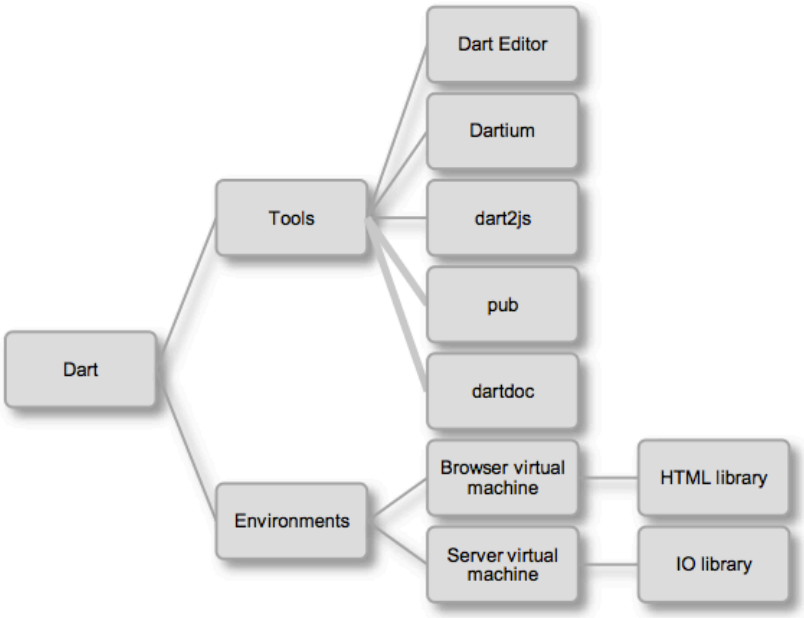

Figura 5 - *Dart também é uma plataforma (Buckett, 2012)*

Em adição ao grande ecossistema de ferramentas que ajudam a construção de aplicações, o Dart é projetado para parecer familiar, seja um programador com conhecimentos server-side, Java e C#, ou client-side, JavaScript e ActionScript.

2.3.1 Por que o Google criou o Dart?

Para Walrath e Ladd (2012), o Google quer que mais e mais pessoas sejam capazes de criar bons web apps. Com mais aplicações web melhorando a web, a web trabalha melhor, todos ganham.

Os engenheiros no Google vem pensando sobre web apps por muito tempo. Já construíram uma boa quantidade de aplicações web complexas e de larga escala (Gmail, Google+, Google Docs, etc), logo eles estão bastante familiares com os desafios de arquitetar aplicações *web*.

Eles também desenvolveram um navegador web (o famoso Chrome) e um motor JavaScript (V8), o que ensinou a eles bastante sobre como rodar aplicações web de forma rápida (alta performance).

Segundo Walrath e Ladd (2012), Dart foi criado porque ele permite a criação fácil e rápida de aplicações *web* complexas, diferente de outras linguagens.

2.3.2 A Web precisa realmente de outra Linguagem?

Desenvolvedores de aplicações de todas as plataformas deveriam ser capazes de construir para a web moderna, porém muitos tem diferentes expectativas de sua linguagem, ferramentas e plataforma de desenvolvimento do que há disponível hoje.

Walrtah e Ladd (2012) acreditam que existem espaço para uma nova plataforma, uma que não esteja coberta por "15 anos de poeira"(JavaScript), que é familiar para desenvolvedores de cenários diferentes, e que é estruturada para permitir grandes, e mais complexas aplicações que tanto são esperadas pelos usuários da web.

Apesar de existir certo conflito com o JavaScript, o mesmo não vai embora tão cedo, e o próprio Google trabalha ativamente para melhorar o JS, trazendo também novas funcionalidades para o navegador Chrome e o motor V8. Porém, isso não impede o pensamento em outras soluções que talvez se mostrem mais eficazes. Por exemplo, o Dart vai aproveitar a continuidade dos trabalhos feitos no JavaScript, pois o mesmo é possível compilar um programa Dart em JavaScript, garantindo a compatibilidade entre toda a *web* moderna.

Strom (2012) acredita que o Dart é uma plataforma familiar que encontra as necessidades de desenvolvedores dos diferentes cenários e áreas, incluindo os mais endêmicos. Dart traz boas ideias para a programação web, e essa inovação só ajuda no avanço da web.

2.3.3 O Que há de novo sobre o Dart?

Como o objetivo do trabalho (informado anteriormente) não é ir muito a fundo na linguagem (nos conceitos comuns entre outras linguagens), serão apresentadas as características mais interessantes.

Para mais detalhes técnicos, o site oficial do Dart (www.dartlang.org) traz a documentação completa.

2.3.3.1 Optional Typing (especificação de tipos opcional)

A utilização de tipos (*types*) ou não fica por conta do desenvolvedor. Os tipos no código em Dart não muda a maneira que a aplicação é executada, mas eles pode ajudar desenvolvedores e ferramentas de programação à entender código mais facilmente.

Tal característica é comum principalmente na prototipação, como maneira rápida e eficiente de escrever o código, porém é interessante adicionar o tipos quando a aplicação for finalizada. Um padrão bastante utilizado, é a utilização dos tipos em assinaturas de métodos e interfaces, mas omitir os mesmos quando dentro de métodos.

2.3.3.2 Snapshots (instantâneos)

Atualmente, os navegadores precisam analisar (*parsing*) o código fonte da aplicação web antes que de sua execução. Dart pode ter uma "instantâneo" próprio (seus estados e códigos gravados em certo ponto no tempo), que aumenta a inicialização consideravelmente. Segundo Buckett (2012), em testes iniciais uma aplicação web com 54.000 linhas de código Dart, iniciou em 640ms sem a utilização da snapshot. Utilizando tal recurso, a mesma aplicação iniciou em 60ms. Quando um programa em Dart está rodando dentro da Dart VM, é possível observar as melhorias na performance facilmente.

2.3.3.3 Isolates

Dart suporta execuções simultâneas com a utilização de *isolates*, vários processos isolados. Cada *isolate* tem sua própria memória e código, os quais não podem ser afetados por outro *isolate*. A única forma de um *isolate* se comunicar com outro *isolate* é através de mensagens. A utilização de *isolates* permite um único aplicativo utilizar a arquitetura multi-core dos computadores modernos. Um outro uso seria a execução de códigos de diferentes origens na mesma página, sem comprometimento na segurança.

2.3.3.4 Generics

A implementação de códigos genéricos já foi feita antes, por outras linguagens, porém sempre foi confuso e difícil. Dart adota uma nova abordagem para utilização de *generics* que é mais fácil para o desenvolvedor entender. Porém existe o sacrífico na exatidão, mas mesmo assim o retorno é grande na produtividade.

2.3.3.5 Biblioteca HTML

Dart possui uma biblioteca nativa para a manipulação direta do DOM (a interface que permite de forma programável atualizar o conteúdo, estrutura, e estilo de uma página web). Segundo Walrath e Ladd (2012), a forma com que o Dart acessa o DOM foi pensada de forma inovadora, tendo elementos, atributos, e nós naturais de se trabalhar.

2.3.4 Por que Dart parece tão familiar?

Dart é projeto para adoção em massa, logo ele tem um estilo familiar tanto para programadores de linguagens script (como JavaScript) como para os de linguagens estruturadas (como Java).

Entretanto, o Dart possui características únicas. Por exemplo, interfaces com implementação padrão, *isolates*, optional typing, e possibilidades de programação no lado do servidor (como node.js).

2.3.5 O que existe dentro da Plataforma Dart?

Como mencionando anteriormente, Dart acabou se tornando mais que apenas uma linguagem, é também uma plataforma completa para desenvolvedores web modernos.

Dentro da plataforma temos:

- **A Linguagem:** A linguagem Dart é familiar, com várias características novas e únicas.

- **Bibliotecas:** Bibliotecas de núcleo provendo várias funcionalidades incluindo coleções, datas, operações matemáticas, HTML, server-side I/O (sockets), e até mesmo JSON.

- **Compilador para JavaScript:** é possível compilar programas Dart para JavaScript, por meio do utilitário dar2js. Dessa forma, problemas de compatibilidade são eliminados facilmente.

- **Dart VM**: a plataforma Dart conta com uma máquina virtual construída do zero para rodar códigos Dart nativamente. A VM roda em linda de comando para aplicações *server-side*, e também pode ser embutida em navegadores para aplicações *client-side*.

- **Dartium**: A Dart VM foi embutida dentro de uma versão do navegador Chromium (versão *open source* do Chrome), apelidado de Dartium, permitindo a execução nativa das aplicações Dart, sem necessidade de conversão para JavaScript.

- **Dart Editor**: um editor de código leve, completo com destaque de sintaxe e ajuda para completar código. Ele permite executar códigos Dart tanto na VM quanto no Dartium. É dentro do editor que está localizado o utilitário dart2js

2.3.6 Bibliotecas

Por hora, o Dart conta com três bibliotecas oficiais:

2.3.6.1 dart:core

Contém as API's básicas que todas as aplicações podem utilizar, sejam elas "*standalone scripts*" ou dentro do navegador. Essas API's permitem as seguintes operações:

- Mostrar textos básicos (*print*())

- Agrupar objetos em coleções (*Collection*, *Set, List, Queue*)
- Manipular pares chave-valor (*Map*)
- Especificar datas e tempos (*Date, TimeZone, Duration, Stopwatch*)
- Utilizar *strings* (*String, Pattern, RegExp*)
- Utilizar números (num, int, *double*, *Math*)
- Retornar valores antes mesmo que a operação esteja completa (*Future*)
- Comparar objetos similares (*Comparable*)
- Realizar uma operação em cada item de um objeto multi-item (*Iterable*)

2.3.6.2 dart:html

Contém API's para produzir as interfaces de usuários para aplicações web. Por exemplo:

- Retornar objetos globais (*document, window*)
- Procurar elementos HTML (*query*() e *queryAll*())
- Adicionar e remover tratamentos de evento (propriedade *on*)
- Operar em grupos de objetos usando as interfaces embutidas de coleção

2.3.6.3 dart:io

Contém as API's server-side, para conexão para o mundo de fora. Essa biblioteca é acessada somente quando a Dart VM está rodando no servidor.

- Ler e escrever dados (*InputStream*, *OutputStream*)
- Abrir e ler arquivos (*File*, *Directory*)
- Conectar em *network sockets* (*Socket*)

2.3.7 Um pouco de código

No capítulo 2.3.3 foram mostradas algumas das características mais interessantes da linguagem Dart. Agora algumas delas serão elaboradas, para demonstração de código.

2.3.7.1 Types

```
class Point
{
 Point(this.x, this.y);
 distanceTo(other)
 {
  var dx = x - other.x;
  var dy = y - other.y;
  return Math.sqrt(dx * dx + dy * dy);
 }

 var x, y;
}

main()
{
 var p = new Point(2, 3);
 var q = new Point(3, 4);
 print('distance from p to q = ${p.distanceTo(q)}');
}
```

O código acima define e faz o uso de uma classe, e a mesma não possui tipos (*types*). Tipos são opcionais no Dart. Eles não mudam a maneira que os programas rodam, mas eles tornam o código mais legível para ferramentas (como debuggers e IDE's) e desenvolvedores (novas pessoas no projeto terão mais facilidade). Tipos são como anotações ou uma documentação que ajudam ferramentas ou pessoas entenderem o código rapidamente e solucionar bugs de forma eficaz.

Um bom lugar para a especificação de tipos é na assinatura de métodos e interfaces. Quanto mais membros dentro de um projeto, e mais classes são criadas, é importante saber quais tipos são usados (e retornados) pelos métodos.

Abaixo o exemplo anterior com os tipos adicionados nas definições de API:

```
class Point
```

```
{
  Point(this.x, this.y);
  num distanceTo(Point other)
  {
    var dx = x - other.x;
    var dy = y - other.y;
    return Math.sqrt(dx * dx + dy * dy);
  }

  num x, y;
}

main()
{
  var p = new Point(2, 3);
  var q = new Point(3, 4);
  print('distance from p to q = ${p.distanceTo(q)}');
}
```

2.3.7.2 Manipulando o DOM

A manipulação do DOM via Dart possui algumas vantagens. É possível a utilização de elementos e nós como se fossem objetos, e percorrer nós-filhos como nas coleções do Dart.

```
#import("dart:html");

main()
{
  // Find an element.
  var elem = document.query("#id");
  // Handle events.
  elem.on.click.add((event) => print('click!'));

  // Set an attribute.
  elem.attributes['name'] = 'value';
  // Add a child element.
  elem.elements.add(new Element.tag("p"));
  // Add a CSS class to each child element.
  elem.elements.forEach((e) => e.classes.add("important"));
}
```

2.3.7.3 Isolates

Apesar do Dart ser single threaded, ainda é possível aproveitar todo o potencial de máquinas multi-core, utilizando *isolates*. Um *isolate* prove a isolação de uma memória entre diferentes partes de um programa em execução.

Cada *isolate* pode rodar e uma *thread* ou processo separado, gerenciado pela Dart VM.

Isolates se comunicam por envio de mensagens através de portas. As mensagens são copiadas para que não ocorra alteração de estado dos objetos em transação.

O exemplo abaixo ilustra a utilização (a função echo é chamada por cada *isolate*):

```
#import('dart:isolate');

echo()
{
  port.receive((msg, reply) => reply.send("Echo: $msg"));
}

void main()
{
  SendPort sendPort = spawnFunction(echo);
  sendPort.call("Hello, Dart!").then((response) => print(response));
}
```

III Desenvolvendo uma Aplicação Web com Dart

Neste capítulo serão apresentadas as ferramentas e tecnologias utilizadas, detalhamentos de código-fonte assim como os principais processos na forma imagens/ilustrações.

3.1 Proposta

Será construído um projeto que irá demonstrar o funcionamento do Dart para aplicações "completas" (cliente-servidor). A aplicação-cliente atuará como um blog, onde o usuário pode realizar postagens, e visualizar as mesmas ordenadas por data.

As postagens são enviadas para uma aplicação-servidor (local), onde são salvas em uma base de dados MongoDB, para consultas futuras.

O projeto não dará ênfase nas questões visuais, mas sim na integração cliente-servidor (juntamente com um *database* NoSQL) utilizando somente o Dart.

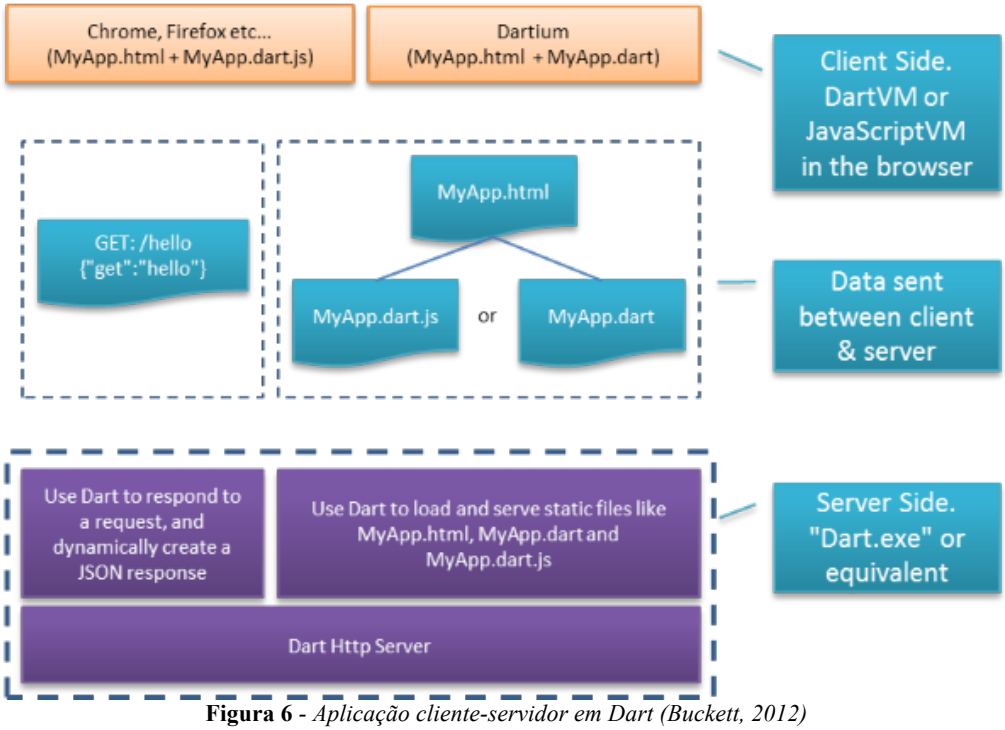

Figura 6 - *Aplicação cliente-servidor em Dart (Buckett, 2012)*

3.2 Ferramentas Utilizadas

Para facilitar o desenvolvimento do projeto, foram utilizadas algumas ferramentas para acelerar processos e dar uma maior qualidade para o código.

3.2.1 Dart Editor

Para Buckett (2012), apesar de ser possível escrever códigos Dart em qualquer editor de texto, o desenvolvedor consegue a melhor experiência com a utilização do Dart Editor.

O Dart Editor foi construído usando o Eclipse RCP, um framework para a construção de editores customizados. Segundo Buckett (2012), dentro do Dart Editor temos as características usuais como *code completion*, navegação e *code outlining*, juntamente com análises estáticas como avisos e erros.

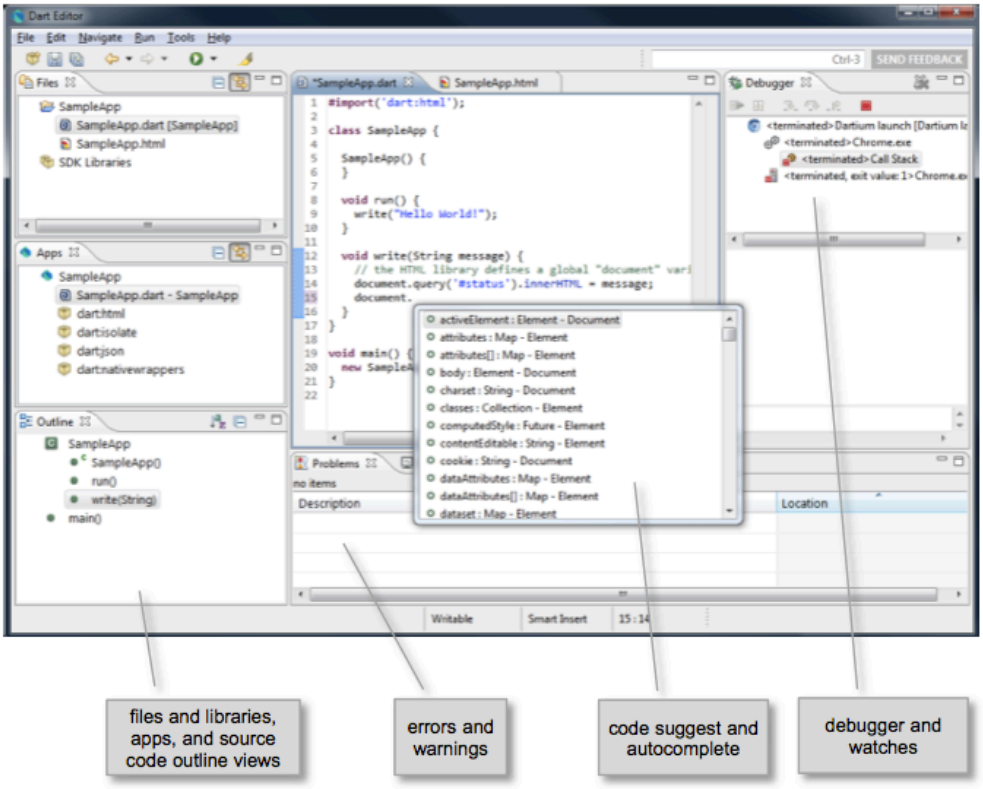

Figura 7 – *O Dart Editor, e algumas funcionalidades (Buckett, 2012)*

Usando o Dart Editor, o desenvolvedor pode escrever códigos associados com uma

página HTML, ter o código Dart convertido para JavaScript e aberto no navegador desejado através da ferramenta *dart2js*.

Caso o código não tenha ligação com páginas HTML, o mesmo vai rodar através do terminal (linha de comando).

3.2.2 Dartium

De acordo com Buckett (2012), o Dartium é um versão customizada do Chromium (versão open source do Google Chrome) com a Dart VM embutida. Ele reconhece scripts do tipo *<application/dart>* e executa os códigos Dart nativamente dentro do navegador sem necessidade de conversão para JavaScript. Ele inclui ferramentas para os desenvolvedores web, facilitando o trabalho de criação para aplicações web ricas.

Juntamente com o Dart Editor, o desenvolvedor consegue um *debugging* rápido e fácil, que permite a validação de valores, variáveis e elementos HTML.

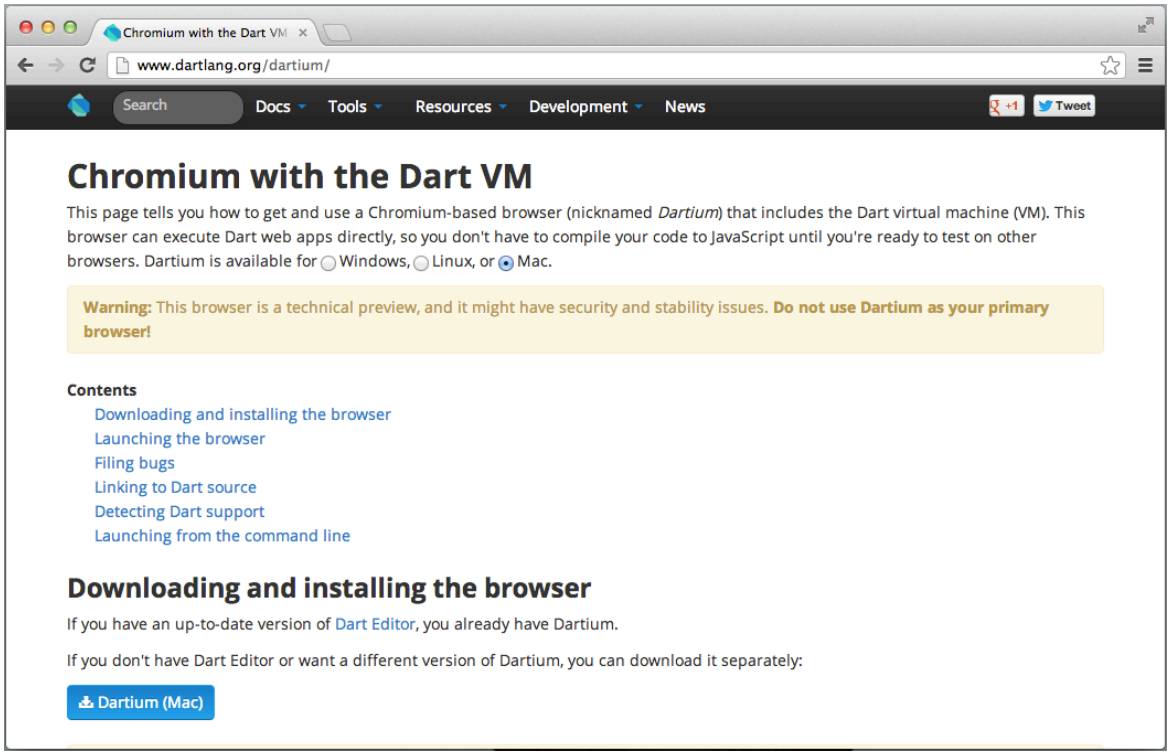

Figura 8 - *Dartium, o navegador customizado para o Dart*

3.2.3 MongoDB

Segundo Maccaw (2011), MongoDB (de "*humongous*") é um *database* NoSQL *open source*, escalável e de alta performance. Ele utiliza documentos no estilo JSON para representar, consultar e modificar dados. Internamente os dados são armazenados em BSON.

Como não segue o padrão de banco de dados relacionais, as informações armazenadas geralmente são versões "serializadas" em JSON de classes do sistema utilizador.

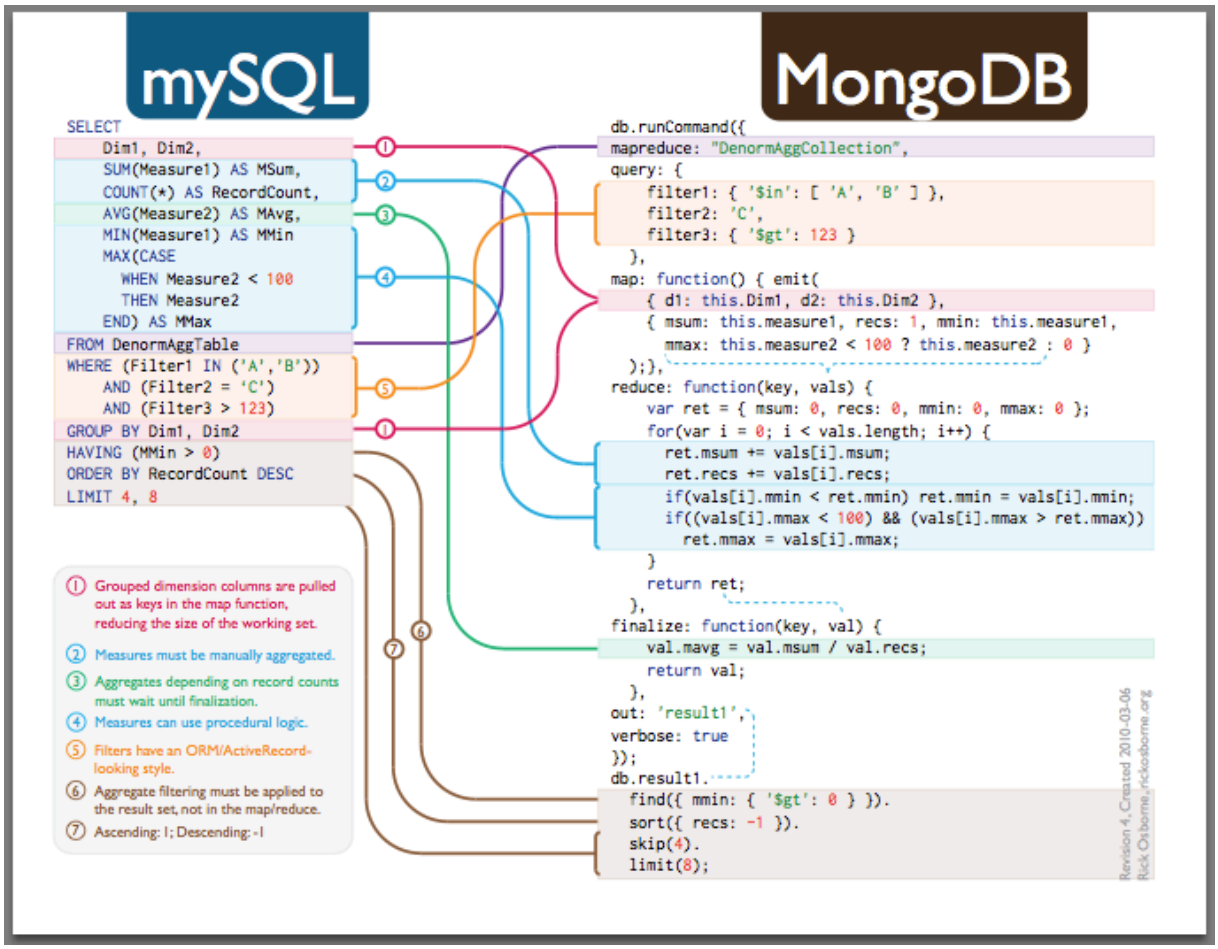

Figura 9 - *Infográfico comparando o mySQL e MongoDB (sysd.org, 2012)*

3.2.4 Bibliotecas

É possível utilizar bibliotecas externas em projetos Dart, para auxiliar no desenvolvimento e trazer funcionalidades específicas. No protótipo desenvolvido foram

utilizadas duas bibliotecas (ambas utilizadas somente no lado do servidor):

- **CrimsonHttp** – estrutura de servidor HTTP extensível para o Dart
- **Mongo-dart** – driver do MongoDB para programação em Dart

3.3 Desenvolvimento do protótipo

Como a aplicação possui dois funcionamentos, os mesmos serão divididos para uma melhor definição.

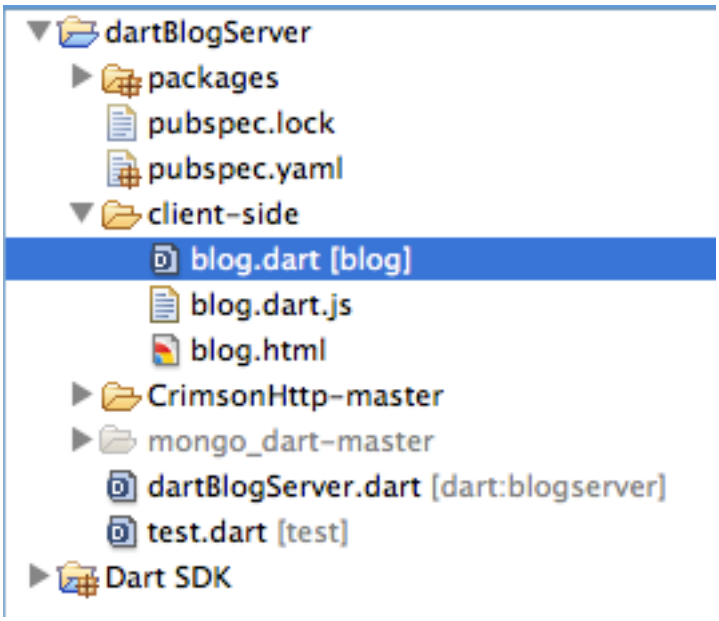

Figura 10 - *Estrutura de arquivos do protótipo*

3.3.1 Server Side

A parte inicial é definida pela integração funcional com a biblioteca *CrimsonHttp,* tendo ela rodando e respondendo para o navegador.

Temos a criação da classe *BlogServer*, que contém o método *run()*, para criar uma instância do *CrimsonHttpServer*, e um módulo para este servidor.

Então são adicionados alguns *handlers* para o módulo, sendo eles os responsáveis por enquadrar as requisições ao servidor de acordo com as ações configuradas.

A parte responsável pela integração com o MongoDB é mais simples. Dentro da classe *BlogServer* foi adicionado uma variável chamada _posts que representará a coleção do MongoDB destinada às postagens do blog.

Foi criado um construtor default, que deve estabelecer a conexão com a base de dados, e abrir a coleção de posts para o blog (caso exista alguma).

Para o funcionamento desta parte, é importante verificar se o serviço instalado do MongoDB está rodando (comando mongod, no caso).

```
class BlogServer {

DbCollection _posts;

 BlogServer() {
   Db db = new Db("mongodb://localhost/test");
   db.open();
   _posts = db.collection("posts");

   // uncomment the following line to remove all the posts at app startup
   //_posts.remove();
 }

 void run()
{
   CrimsonHttpServer server = new CrimsonHttpServer();
   CrimsonModule module = new CrimsonModule(server);
   module.handlers
            //.addEndpoint(new Favicon())
            .addFilter(new CookieSession())
            .addEndpoint(new Route("/post/all", "GET", getPosts))  //return all posts
            //.addEndpoint(new Route("/post", "GET", getRecentPost))  //return the first post
            .addEndpoint(new Route("/post", "POST", addPost)) //add a post
            .addEndpoint(new StaticFile("./client"));

   server.modules["*"] = module;  //this is the default module.

   server.listen("127.0.0.1", 8083);
}
```

Como descrito antes, o MongoDB realiza seu armazenamento dos dados em BSON. Isso significa que é possível a leitura e escrita de Maps utilizando JSON convertido.

3.3.1.1 Lendo e escrevendo informações

Os *Route Handlers* obrigam o uso de um método que receba um objeto *httpRequest*,

httpResponse e *CrimsonData*. Ele também deve retornar um *Future*, ou um valor nulo.

A informação enviada do navegador para o servidor vai no formato *StringBuffer*, que é lido na variável *postData*, para utilização do conversor JSON.*parse*, convertendo de uma *string* simples para um Map.

Após estes processos, é possível a chamada da função de inserção no MongoDB *(_posts.insert()*).

```
Future addPost(HttpRequest req,res,data)
{
  print("adding post: ${req}");
  Completer completer = new Completer();

  StringBuffer body = new StringBuffer();
  StringInputStream input = new StringInputStream(req.inputStream);
  input.onData = () => body.add(input.read());
  input.onClosed = () {
    String postdata = body.toString();
    print("postdata: ${postdata}");
    if (postdata != null) {

      Map blogPost = JSON.parse(postdata);
      print(blogPost);
      Map savedBlogPost = new Map<String,Object>();
      savedBlogPost["text"] = blogPost["text"];
      savedBlogPost["posted"] = blogPost["posted"];
      _posts.insert(savedBlogPost);
      completer.complete(data);
    }
  };

  return completer.future;
}
```

O último método para o funcionamento parcial do servidor, é o *getPosts*. Responsável por trazer a coleção de postagens armazenadas no *database*. Elas são convertidas para JSON, permitindo uma iteração entre os membros da cadeia de *string*, e enviando para o formulário.

```
Future getPosts(req,res, CrimsonModule data) {
  Completer completer = new Completer();

  List postList = new List();

  Cursor cursor = _posts.find({});

  cursor.each((Map post) {
```

```
    print("is the post a map?${post is Map}");

var id = post["_id"];
    post.remove("_id");
    post["_id"] = id.toString();

     postList.add(post);

  }).then((dummy) {
        print(postList);
    try {
      String postAsString = JSON.stringify(postList);
      res.outputStream.writeString(postAsString);
      completer.complete(data);
    }
    catch (ex, stack) {
      print(ex);
      print(stack);
      completer.completeException(ex);
    }
  });

  return completer.future;
 }
```

Até este momento, somente a parte servidor está funcional, faltando a interface para o usuário interagir (armazenando postagens no *blog*).

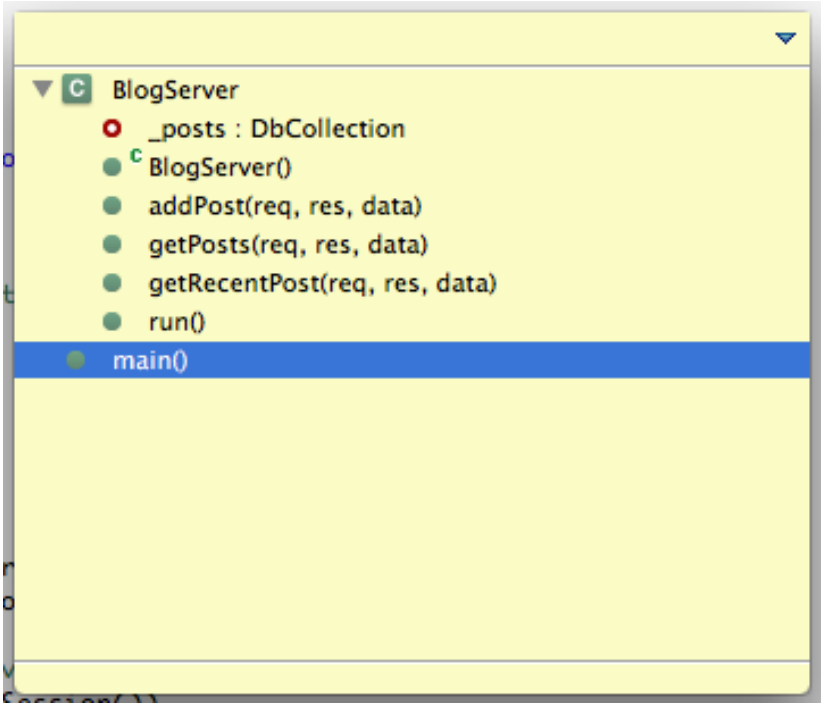

Figura 11 - *Membros da class BlogServer*

3.3.2 Client Side

A parte “cliente” será baseada em um arquivo .html e um arquivo .dart. Não existirá processamento nesse momento, somente o envio de informações para a interface configurada com o servidor.

A construção da interface é simples, e não haverá efeitos especiais nem CSS, pois será controlada diretamente pelo código Dart (utilizando a biblioteca dart:html, para acesso ao DOM).

O Dart Editor foi um grande facilitador, criando os arquivos inicias automaticamente via o “New Web Application Wizard“.

```
<!DOCTYPE html>
<html>
 <head>
  <title>blog</title>
 </head>
 <body>
  <style>
   #textbox { width:150px; height:100px; float:right;  }
   #add-post-button { float:right; }
   #list-of-posts { width: 350px; float:left; }
  </style>
  <script type="application/dart" src="blog.dart"></script>
  <script src="http://dart.googlecode.com/svn/branches/bleeding_edge/dart/client/dart.js"></script>
 </body>
</html>
```

A página deveria funcionar em vários navegadores, por isso teve seu código Dart convertido para JavaScript, e o mesmo incluso no projeto. Logo, a aplicação não fica amarrada ao Chromium, para ter sua execução feita corretamente.

A interface contém um *html textarea*, um *html button* e uma *div* para acomodar a lista de postagens feitas pelo usuário. Cada elemento contém um id, para poderem ser acessados pelo código Dart.

```
void buildUi()
{
 Element input = new Element.tag("textarea");
 input.id = "textbox";
 document.body.nodes.add(input);

 var button = new Element.tag("button");
```

```
  button.text="Add Post";
  button.id="add-post-button";
  button.on.click.add(_onAddPostClick);
  document.body.nodes.add(button);

  var postDiv = new Element.tag("div");
  postDiv.id = "list-of-posts";
  document.body.nodes.add(postDiv);
}
```

Com a interface ajustada, deve-se carregar as informações do servidor. Isso é feito utilizando requisições HTTP, através do XMLHttpRequest que envia uma solicitação para o endereço configurado nas rotas (http://localhost:8083/post/all).

Como já existe uma rota configurada para o endereço, sempre que ele for acessado haverá uma consulta no *database* MongoDB, e o servidor retornará um JSON das informações.

```
void loadExistingPosts() {
//load the posts
  var req = new HttpRequest.get("http://127.0.0.1:8083/post/all", (result) {
    List posts = JSON.parse(result.responseText);
    for (Map blogPost in posts) {
      //add each post back to the UI
      addPostToUi(blogPost);
    }
  });

            }
```

E assim temos um protótipo realizando a troca de informações entre servidor e cliente, com acesso ao *database* NoSQL desejado.

Interessante apontar alguns conceito da Web 2.0, utilizados neste projeto:

- Apresentação de conteúdo dinâmico
- Troca de informações assíncronas
- Manipulação de dados, e total dependência dos mesmos para um resultado
- ~~JavaScript~~ Dart ligando tudo

3.4 Demonstração

A seguir, algumas telas da aplicação web (protótipo) e detalhes, para demonstrar seu funcionamento.

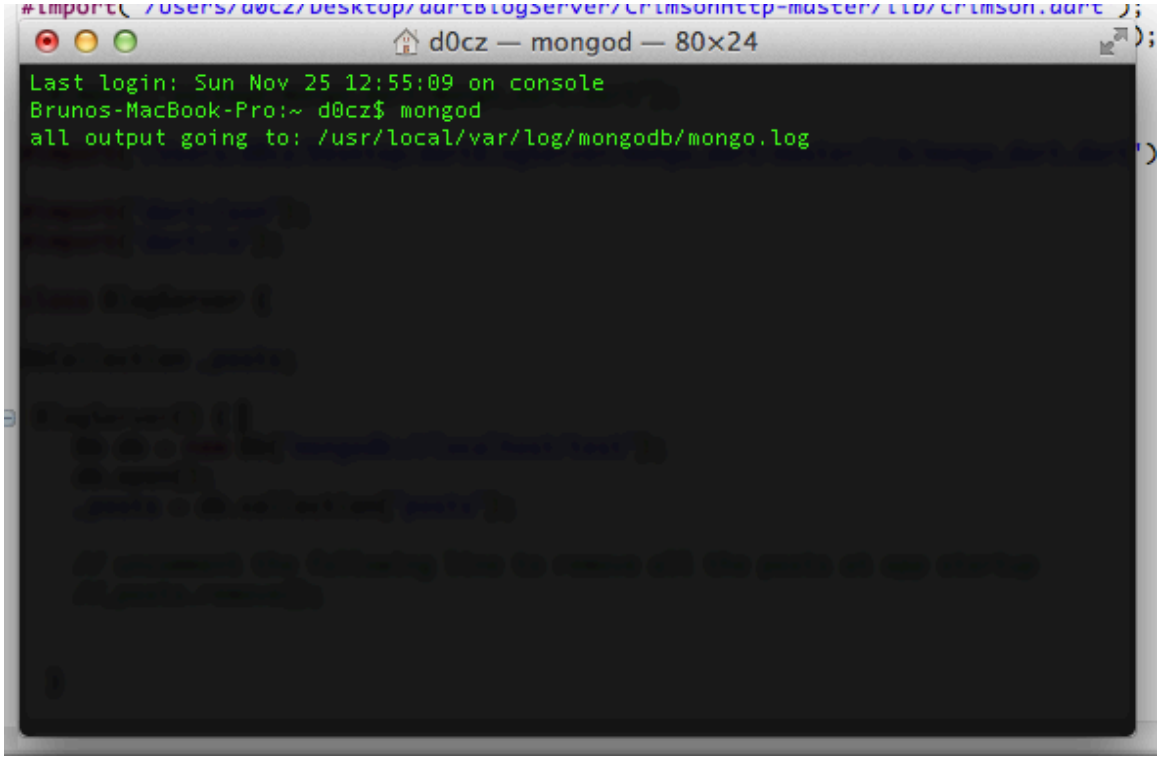

Figura 12 - *Detalhe: inicialização do serviço MongoDB*

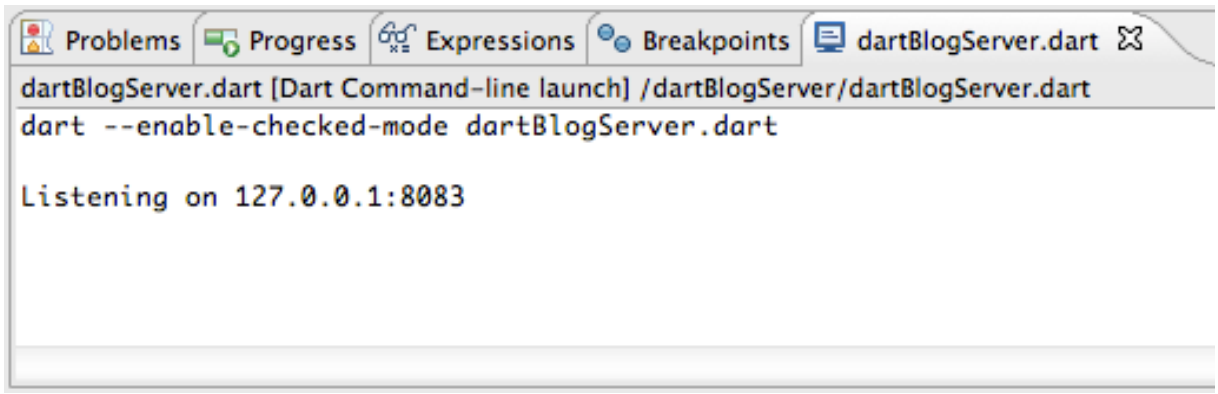

Figura 13 - *Inicialização do Web Server do protótipo (terminal)*

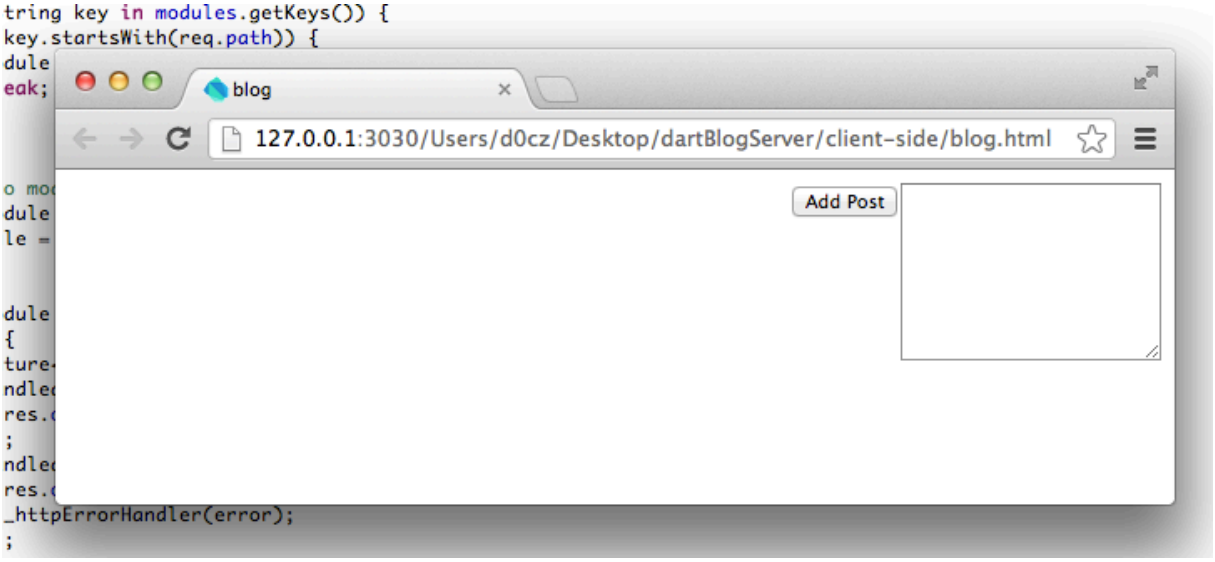

Figura 14 - *Tela inicial do protótipo, sem posts antigos*

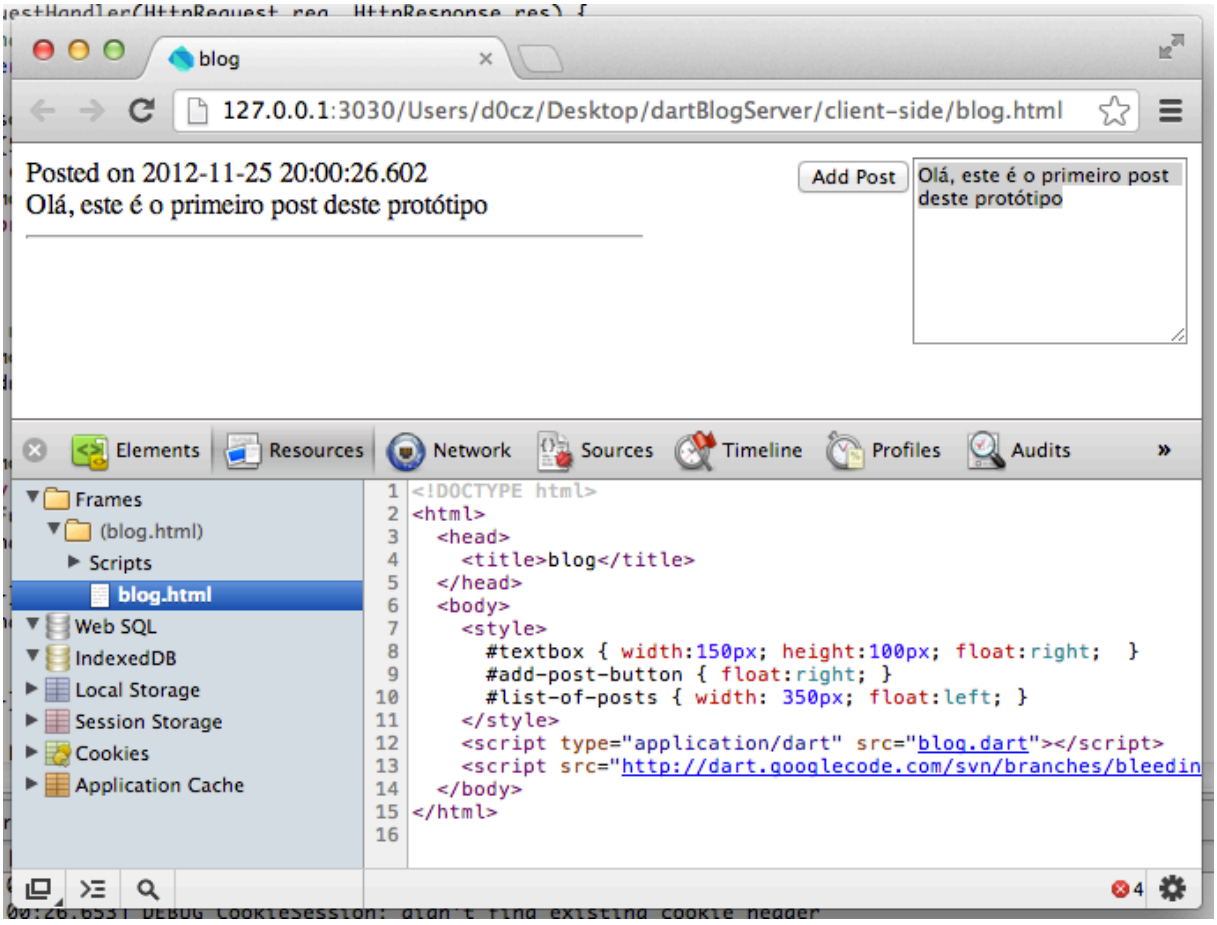

Figura 15 - *Realização de um post, onde o mesmo é armazenado*

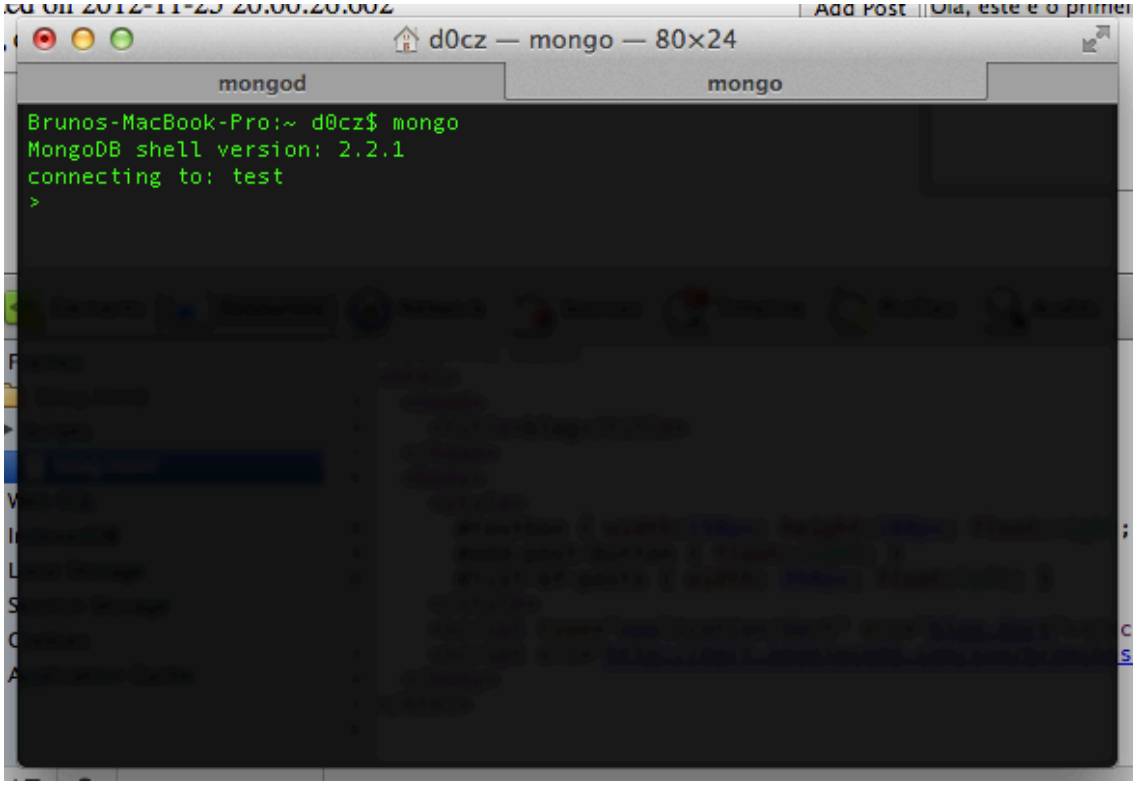

Figura 16 - *Console de utilização do MongoDB*

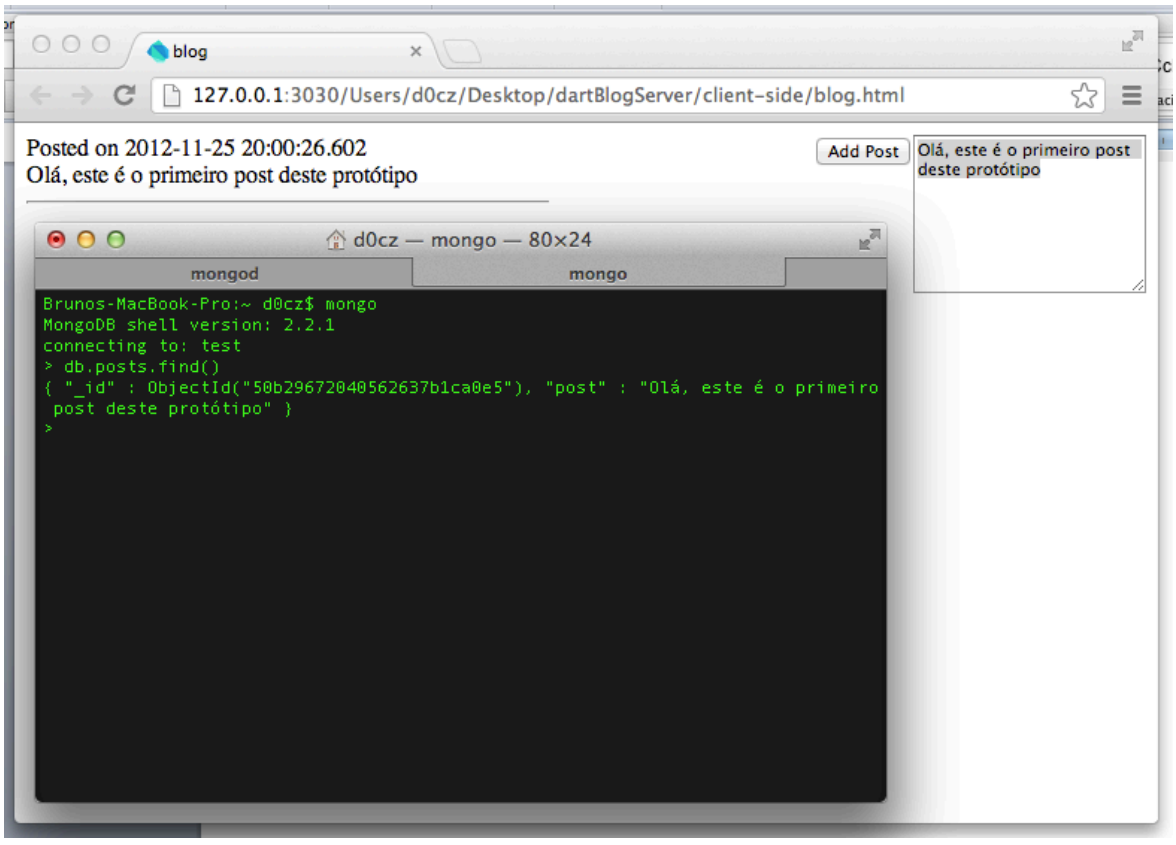

Figura 17 - *Validação do sucesso para o post*

IV CONCLUSÃO

Este trabalho apresentou um estudo sobre a web moderna, desenvolvimento de aplicações web e uma visão inicial da plataforma Dart. Com a realização destes estudos, e a construção de um simples protótipo, foi possível conhecer uma área da computação que não é muito focada pelo curso.

Ficou claro que o desenvolvimento de aplicações web é o futuro, onde cada dia que passa dependemos basicamente de uma só aplicação desktop: o navegador. Através dele, pode-se consumir qualquer tipo de conteúdo, de forma rápida e interativa. “O sistema é a rede”, então um sistema é tão bom quanto a rede que o conecta. E um sistema autônomo (não conectado) é tão bom quanto um peso de papel.

Com tais possibilidade de conexão e integração emergindo, e a colaboração entre usuários sendo enfatizada, aplicações web são o futuro dominante.

Durante o desenvolvimento da aplicação protótipo, foi possível tirar uma conclusão real sobre a plataforma Dart. A linguagem se mostrou realmente fácil e familiar, e pode ter um futuro interessante. Porém, foi sentida uma dificuldade de utilização em relação ao Dart Editor. Programadores acostumados com IDEs mais ricas e funcionais vão perceber a fraqueza da ferramenta.

Outro ponto interessante de conclusão, é o fato do Dart não contar com uma comunidade grande e ativa, gerando um certo desconforto para iniciantes e até mesmo veteranos de plataformas mais sólidas. Ocorreram dúvidas de programação e utilização das bibliotecas que foram resolvidas após muita pesquisa, o que quebra o conceito de produtividade proposto pela plataforma.

Gostaria também de reforçar que o projeto protótipo apresentado, em nenhum momento teve questões visuais como objetivo, e talvez o mesmo possa parecer “pobre” em relação às expectativas. Mas, garanto que o objetivo real do trabalho foi alcançado: uma aplicação web completa (cliente-servidor), utilizando informações armazenadas em um banco de dados.

V REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, Bryan. **Web 2.0:** a new wave of innovation for teaching and learning? San Francisco: EDUCAUSE Review, vol. 41, no. 2 (March/April 2006): 32–44.

BRODKIN, Jon. **Web 2.0:** buzzword or internet revolution? Disponível em: <http://goo.gl/RwKFy> Acesso em: 20 out. 2012.

BUCKETT, Chris. **Dart in action.** New York: Manning Publications. 2012

FOWLER, Susan & STANWICK, Victor. **Web application design handbook:** best practices for web-based software. San Francisco: Elsevier, 2004.

GARRET, Jesse James. **Ajax:** a new approach to web applications. Disponível em: <http://goo.gl/KtmMD> Acesso em: 20 out. 2012.

MACCAW, Alex. **JavaScript web applications.** California: O'Reilly Media, Inc. 2011.

O'REILLY, Tim. **What is web 2.0?** Disponível em: <http://goo.gl/iNZ0R> Acesso em: 01 nov. 2012.

SKRENTA, Rich. **The incremental web**. Disponível em <http://goo.gl/nN7em> Acesso em: 20 out. 2012.

STROM, Chris. **Dart for hipsters (Fast, Flexible, Structured Code for the Modern Web).** New York: The Pragmatic Bookshelf, LLC. 2012.

WALRATH, Kathy & LADD, Seth. **What is dart?** California: O'Reilly Media, Inc. 2012.

ZALEWSKI, Michal. **The tangled web (A Guide to Securing Modern Web Applications)** San Francisco: No Starch Press, 2012.