UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

RODOLFO SEFFRIN

COMPARATIVO ENTRE OS *FRAMEWORKS* DE DESENVOLVIMENTO *WEB* VAADIN E GWT UTILIZANDO A BIBLIOTECA SMARTGWT.

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2013

RODOLFO SEFFRIN

COMPARATIVO ENTRE OS *FRAMEWORKS* DE DESENVOLVIMENTO *WEB* VAADIN E UTILIZANDO A BIBLIOTECA SMARTGWT.

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof Fernando Schütz, M.Sc.

MEDIANEIRA

2013

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

COMPARATIVO ENTRE OS *FRAMEWORKS* DE DESENVOLVIMENTO *WEB* VAADIN E GWT UTILIZANDO A BIBLIOTECA SMARTGWT.

Por

Rodolfo Seffrin

Este Trabalho de Diplomação (TD) foi apresentado às 08:30 h do dia 27 de março de 2013 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. Os acadêmicos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. Me. Fernando Schütz
UTFPR – Câmpus Medianeira
(Orientador)

Prof. Me. Everton Coimbra de Araujo
UTFPR – Câmpus Medianeira
(Convidado)

Prof. Me. Juliano Rodrigo Lamb
UTFPR – Câmpus Medianeira
(Convidado)

Prof. Me. Juliano Rodrigo Lamb
UTFPR – Câmpus Medianeira
(Responsável pelas atividades de TCC)

RESUMO

SEFFRIN, Rodolfo. Comparativo entre os *frameworks* de desenvolvimento *Web* Vaadin e GWT Utilizando a biblioteca SmartGWT. Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. Universidade Tecnológica Federal do Paraná. Medianeira, 2012.

Com o crescimento da Internet nos últimos anos e aplicativos *Web* cada vez mais robustos, surgiu à necessidade da utilização de *frameworks* que auxiliam no desenvolvimento de aplicações *Web* RIA (Rich Internet Application). Assim, o presente trabalho realizou uma análise comparativa entre dois *frameworks* RIA de desenvolvimento *Web*: Vaadin e GWT (Google *Web* Toolkit). Com GWT foi utilizado a biblioteca de componentes SmartGWT para auxilio na construção da *interface* gráfica. Tanto Vaadin como GWT utilizam como elemento principal para desenvolvimento da aplicação a linguagem Java, e se assemelham muito com o estilo de desenvolvimento *desktop*. Para demonstrar o funcionamento de ambos foi desenvolvida uma aplicação para acompanhamento no crescimento de uma criança, em que o ciclo inicia no seu nascimento, terminando quando sua fase de crescimento inicial estiver concluída. Os resultados da avaliação concluíram que o *Framework* Vaadin é o mais indicado para a utilização, devido a sua arquitetura de programação ser produtiva e ter componentes agradáveis para o usuário final.

Palavras-chaves: RIA, Aplicações *Webs*, Java.

RESUMO EM LINGUA ESTRANGEIRA

SEFFRIN, Rodolfo. Comparative between *Web frameworks* Vaadin and GWT Using The Library SmartGWT. Trabalho de Conclusão do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. Universidade Tecnológica Federal do Paraná. Medianeira, 2012.

With the growth of the Internet in recent years, Web applications and increasingly robust, the need arose the use of frameworks that help in developing web applications RIA (Rich Internet Application)., This study conducted a comparative analysis between two frameworks RIA Web Development: Vaadin and GWT (Google Web Toolkit). With was used SmartGWT component library for assistance in building the graphical interface. Vaadin uses GWT as far as the main element for application development language Java, and closely resemble the style of desktop development. To demonstrate the operation of both an application was developed for monitoring the growth of a child, where the cycle begins at birth, when ending its initial growth phase is complete. The evaluation results indicated that the Vaadin Framework is the most appropriate for use, due to its programming architecture components to be productive and have enjoyable for the end user.

Keywords: RIA, Applications *Webs*, Java.

LISTAS DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

O crescimento da *Web* nos últimos anos criou a necessidade de serviços mais rápidos e instantâneos, além disso, as aplicações desenvolvidas estão se tornando mais complexas e ágeis devido as suas novas características de negócio. Com este crescimento surgiu o conceito *Web* 2.0. Segundo a definição de O'Reilly (2004), Web 2.0 é

> "*Web* 2.0 é 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 de rede para se tornarem melhores quanto mais são usados pelas pessoas, aproveitando a inteligência coletiva".

E uma das principais características desta nova mudança é que as paginas *Web*s passaram a se tornar interativas, e com a evolução dos equipamentos de acesso a Internet, torna-se possível o processamento de certas informações no próprio cliente, fazendo com que a comunicação de requisição e resposta seja assíncrona, ou seja, não é mais necessário atualizar toda a aplicação para refletir uma pequena atualização na página. (GONÇALVES e VAHL, 2010)

Por Meio disto, as aplicações *Webs* passaram a ter as mesmas funcionalidades que as *desktop*. Surge então o conceito de RIA (*Rich Internet Application*), que são Aplicações Ricas para Internet que descreve um novo conceito de interface para as aplicações *Web*. Macromedia, (2003), descreve algumas das principais características do termo RIA:

- Alta performance em tempo de execução, conteúdo e comunicação, fazendo com que o conteúdo se torne mais dinâmico, pois a comunicação entre o cliente e o servidor acontece de forma assíncrona, sem precisar atualizar toda a página para se obter uma atualização.
- Integração, em um mesmo ambiente de interface, comunicação e conteúdo. Em aplicações *Web* 1.0 era necessário baixar o conteúdo para a máquina cliente, sendo que na *Web* 2.0 é executado na própria página HTML, utilizando *plugins*[1] para o navegador.
- Multiplataforma: as aplicações tem que ser desenvolvidas para suportar interfaces de celulares, *podcasts* e smartphones, por exemplo.

[1] Segundo PRADA, (2008) plug-in é "todo programa, ferramenta ou extensão que se encaixa a outro programa principal para adicionar mais funções e recursos a ele".

Assim, muitos *scripts*[2] foram desenvolvidos para serem executados no lado cliente, no intuito de melhorar o processamento e deixar mais ágeis os sistemas *Webs*. Outro fator que facilitou no desenvolvimento de aplicações Webs RIA, por utilizarem o melhor do conceito da WEB 2.0, foram os *frameworks*. (GONÇALVES e VAHL, 2010)

Existem inúmeros *frameworks* para serem usados na criação de aplicações *Webs*, a maioria com o objetivo de abstrair trabalhos repetidos, como a criação de componentes e desenvolvimento da parte gráfica da aplicação. Outro exemplo é a facilidade de aplicar padrões de projeto, pois o *framework* já oferece uma estrutura pré-desenvolvida para isso. Para a linguagem Java existem diversos *Frameworks* com este intuito, alguns exemplos são: JSF (Java Server Faces), PrimeFaces, RichFaces, VRaptor, Vaadin, GWT (Google Web Toolkit). A diferença entre eles é o modo como trabalham. Por exemplo, Vaadin e GWT utilizam um conceito diferente que os demais *Frameworks* citados, sua arquitetura de programação é baseada apenas na linguagem Java, não utilizam a maneira tradicional de desenvolvimento de aplicações *Webs*, se assemelham mais ao estilo de desenvolvimento *Desktop*.

O *framework* Vaadin já vem com uma estrutura diferenciada de componentes para serem utilizados no desenvolvimento da aplicação. GWT oferece varias extensões para serem usadas, como: SmartGWT, GXT, GWT-EXT, todas com o mesmo objetivo oferecer componentes para integrar na aplicação. No presente trabalho foi utilizado a biblioteca SmartGWT que é uma estrutura baseada em GWT que oferece diversos recursos para facilitar e aperfeiçoar o processo de desenvolvimento. Assim esse trabalho tem o intuito de fazer uma análise comparativa entre os *frameworks* Vaadin e GWT. Nesta comparação pretende-se destacar os principais elemento que diferenciam o desenvolvimento da aplicação em um e outro *framework*, abstraindo assim o melhor de cada um.

1.1 OBJETIVO GERAL

Comparar os *frameworks* Vaadin e GWT (Google *Web* Toolkit), utilizando como estudo experimental uma aplicação para acompanhar o desenvolvimento/crescimento de uma criança.

[2] Terra, (2007) Script é uma lista de comandos que podem ser executados sem interação do usuário;

1.2 OBJETIVOS ESPECÍFICOS

- Realizar um estudo bibliográfico das tecnologias de desenvolvimento para *Web*, *frameworks* e linguagens a serem utilizadas;
- Modelar e desenvolver uma aplicação com o objetivo de controlar todo o histórico de desenvolvimento de uma criança, com os *frameworks* Vaadin e GWT (Google *Web* Toolkit);
- Elaborar o comparativo com as vantagens e desvantagens da utilização desses *frameworks*, por intermédio de testes na aplicação e demonstrativo de códigos;

1.3 JUSTIFICATIVA

Para acompanhar a evolução dos sistemas *Web*, diversas técnicas e tecnologias foram desenvolvidas nos últimos anos. Alguns exemplos são: HTML versão 5, XHTML, CSS versão 3, XPath, Javascript, JQuery, JSF, JSP, Facelets, JSON, Ajax. Portanto o desenvolvedor tem acesso a diversos processos de desenvolvimento, porém precisa realizar, muitas vezes, o trabalho de vários profissionais, pois envolve implementação de linguagens e tecnologias diferentes.

Este trabalho pretende analisar quais são as vantagens de utilizar um *framework* com conceito diferente no desenvolvimento de uma aplicação Web. Vaadin e GWT se assemelham na maneira como trabalham, utilizam apenas a linguagem Java para a construção de toda a aplicação, por este motivo foram escolhidos para a comparação.

E como parâmetro para o comparativo será desenvolvida uma aplicação que tem o intuito de controlar o desenvolvimento de uma criança. Nesta aplicação o médico informara para o sistema o que foi abstraído da consulta, assim sendo terá o controle de todas as informações das consultas que foram realizadas pela criança, no qual poderá observar como foi sua evolução, terá a opção de gerar gráficos para saber como foi seu desenvolvimento no seu nascimento até há ultima consulta realizada, podendo então ter informações importantes durante o período de consultas que a criança realizou.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho está dividido em cinco capítulos:

- No primeiro capítulo será abordada uma introdução e justificativa sobre o assunto do trabalho proposto, além de serem especificados os objetivos específicos e gerais do projeto.
- No segundo capitulo é abordado o assunto por meio de um referencial teórico.
- No terceiro capitulo é feita a documentação da aplicação a ser desenvolvida, como: requisitos funcionais, diagramas e estrutura do trabalho.
- No quarto capitulo será abordado à comparação entre os dois *frameworks* estudados.
- O quinto capitulo contém as considerações finais do trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo será abordado o referencial teórico das tecnologias utilizadas no trabalho.

2.1 APLICAÇÕES *WEB*

Inicialmente, a Internet era uma plataforma para que os usuários pudessem compartilhar informações. Por exemplo, publicar documentos e por meio de hiperlinks[3] fazer referência cruzada entre eles. A partir do momento que a *Web* foi se popularizando, os usuários começaram a ter softwares em suas máquinas que permitiam interagir com toda esta estrutura. (SMEETS, BONESS e BANKRAS, 2009)

Com sua popularização, a *Web* deixou de apenas usar documentos HTML estáticos, e permitiu que fossem disponibilizados documentos dinâmicos, assim sendo nasceu o conceito de aplicação *Web*. Uma Aplicação *Web* é a que está disponível em um servidor central e oferece serviços para qualquer pessoa que tenha acesso à Internet. (SMEETS, BONESS e BANKRAS, 2009)

Com a Internet disponibilizando serviços robustos, as aplicações passaram a terem a necessidade de desenvolver aplicações RIA (Rich Internet Application). A razão disso, é o fato de dependerem de interações assíncronas com o servidor. Por exemplo, uma aplicação do tipo planilha para *Web* se torna um problema se cada vez que o usuário modificar dados em uma célula tenha que carregar toda a aplicação para refletir o calculo efetuado. Com isto a Internet ganhou um novo termo, chamado de *WEB* 2.0. (SMEETS, BONESS e BANKRAS, 2009)

Segundo GONÇALVES E VAHL (2010) apud O'Relly, algumas características que marcam os aplicativos na *WEB* 2.0 podem ser tidas como:

- *Web* Como Plataforma: A *WEB* passou a ser vista como uma plataforma e não apenas como meio de comunicação cliente e servidor. Ou seja, qualquer

[3] Segundo Google (2010) "Um hiperlink é um link para uma publicação. Quando clica no link, o destino vinculado é aberto. O destino é geralmente outra página da Web, mas também pode ser uma imagem, um endereço de email ou um programa. O hiperlink propriamente dito pode ser um texto, uma imagem ou uma forma. Hiperlink ou hiperligação é qualquer elemento de um hiper texto (páginas web) que façam referência a outro texto ou a outra parte deste texto. Ou seja, qualquer área clicável de uma página web é um hiperlink."

equipamento que tenha acesso a Internet pode acessar uma aplicação *WEB*, independente de sistema operacional;

- Uso da Inteligência Coletiva: Os Usuários passaram a fazer parte das aplicações *Webs*. Um exemplo disso é a *Wikipédia*, uma enciclopédia online onde qualquer pessoa pode colocar e editar algum conteúdo;
- Beta perpétuo: Os softwares passam a ser disponibilizados como serviços e não mais como versões. Ou seja, não e mais preciso instalar um software na máquina local. Basta acessar algum serviço na Internet;
- Modelo de programação mais leve: As aplicações devem ser simples de ser utilizadas;
- Experiências ricas do usuário: Com a evolução dos equipamentos de acesso a Internet, parte do processamento das aplicações pode ser feito no próprio cliente. O modelo de comunicação das aplicações passou a ser assíncrona, não sendo necessário atualizar toda a aplicação para refletir uma pequena atualização. As páginas se assemelham com as aplicações *Desktop*;

Uma das tecnologias que contribuíram para que o modelo de comunicação cliente e servidor passassem a ser assíncrono, com maior interação entre o usuário e a página, tornado as aplicações Webs parecidas com as *Desktop*, foi o conjunto de tecnologias AJAX (Asynchronous Javascript and XML). Para CAELUM (2011)

> "AJAX é um conjunto de técnicas de desenvolvimento *Web* para executar tarefas do lado do cliente, por exemplo, modificar pedaços de uma página sem ter que carregar ela inteira. Na verdade o mecanismo é muito simples. De acordo com alguma ação um Javascript envia uma requisição ao servidor como se fosse em background. Na resposta dessa requisição vem um XML que o Javascript processa e modifica a página segundo essa resposta".

A diferença entre as aplicações AJAX com as que não fazem o seu uso, pode ser ilustrada na Figura 1, o cliente faz uma requisição, o servidor processa essa requisição e devolve o resultado para a página, sendo então disponibilizada para o usuário. O problema disso é que o usuário só pode fazer um número limitado de interações com a página, ou seja, ele terá que esperar um tempo até o servidor devolver a resposta, nesse tempo, o usuário não pode interagir com a aplicação, este é o chamado modelo síncrono de interação com a página.

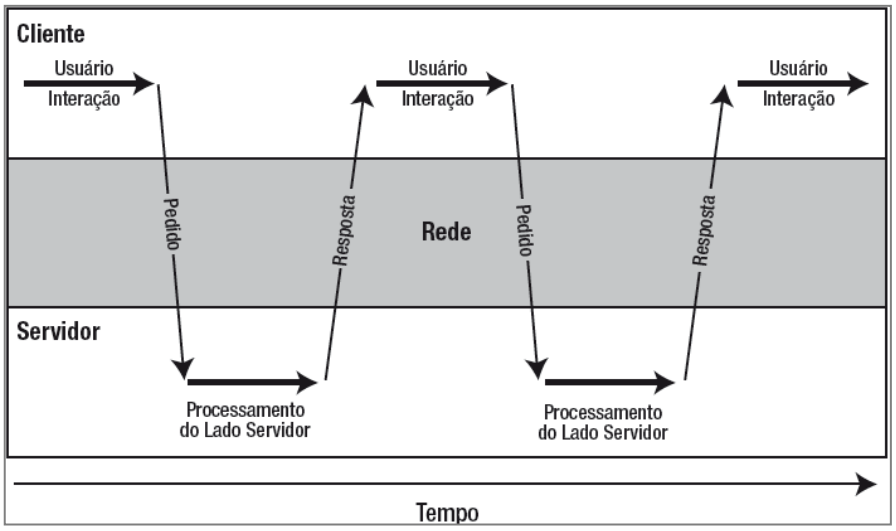


Figura 1 - Modelo de interação de uma aplicação *Web* típica
Fonte: (SMEETS, BONESS e BANKRAS, 2009)

Dependendo do tipo da aplicação, utilizar este modelo de programação é inviável. É nesse sentido que entra o modelo de programação assíncrono, permitindo que o usuário possa interagir com a aplicação de maneira agradável, enquanto alguma ação feita anteriormente é processada no servidor, este modelo pode ser visto na Figura 2.

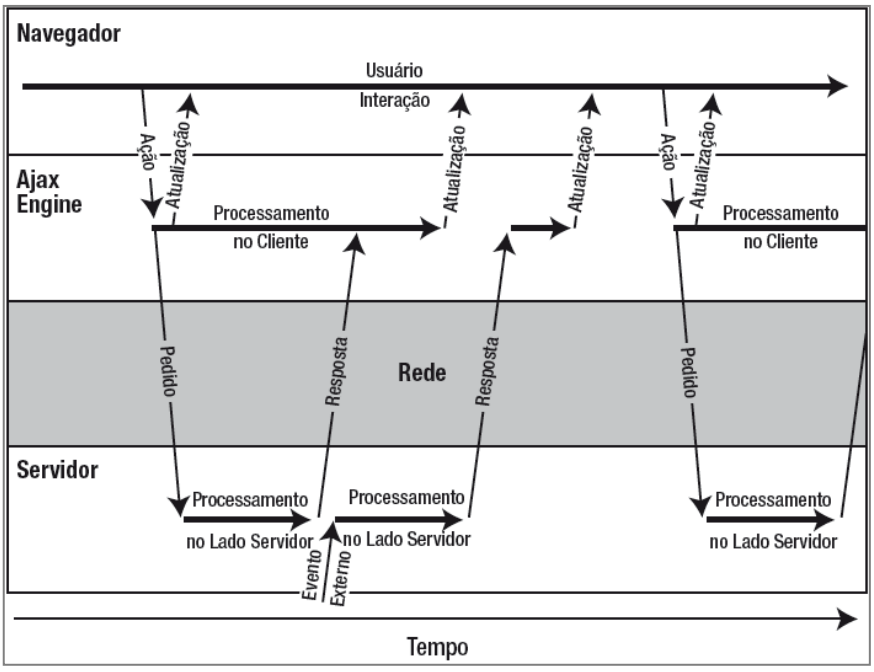


Figura 2 - Modelo de aplicação *Web* utilizando AJAX
Fonte: (SMEETS, BONESS e BANKRAS, 2009)

Com este novo modelo de programação foi quebrado o modelo clássico da *Web* de solicitação da página por HTTP pelo HTML. A maneira que é feito isso é com o *AJAX engine*, que é uma camada de interação entre o usuário e o servidor, isso tudo roda dentro do

navegador e gerencia as ações ao servidor enquanto realiza a manipulação dos resultados, a grande vantagem disso tudo é que o usuário pode continuar interagindo com a aplicação enquanto o servidor processa as informações. (SMEETS, BONESS e BANKRAS, 2009)

2.2 FRAMEWORKS

Um dos pontos principais da Engenharia de Software é o reuso, pois através do uso desta técnica é adquirida maior qualidade e redução do esforço no desenvolvimento (Gimenes & Huzita, 2005). Pelo fato da orientação a objetos fazer com que as classes possam ser reutilizadas, assim como os métodos por meio de herança e polimorfismo (Lundberg & Mattsson 1996). Componentes de softwares fazem com que unidades reutilizáveis sejam definidas para oferecer serviços através de interfaces bem definidas (Gimenes & Huzita, 2005).

Além da reutilização, a tecnologia de *frameworks* torna possível que os produtos sejam gerados através de uma estrutura que utiliza os conceitos mais conhecidos da família de aplicações (Pinto, 2000).

Neste capitulo serão contextualizadas as principais definições de *frameworks* encontrados na literatura. Também será abordado todo o processo de construção de um *framework* e a classificação em que são divididos, além das vantagens e desvantagens da sua utilização.

2.2.1 Definição

A necessidade de desenvolver aplicações cada vez mais robustas e a exigência de produtos com qualidade e confiança, fez com que o processo de desenvolvimento de software passou a utilizar estruturas já desenvolvidas, os *frameworks*.

O objetivo principal de um *framework* é auxiliar em todo o processo de desenvolvimento de um *software*, fazendo com que as aplicações sejam produzidas de forma mais rápida e ágil. Na Figura 3, um exemplo típico da arquitetura de um *framework* para aplicações *Web*. (MOTA, 2010)

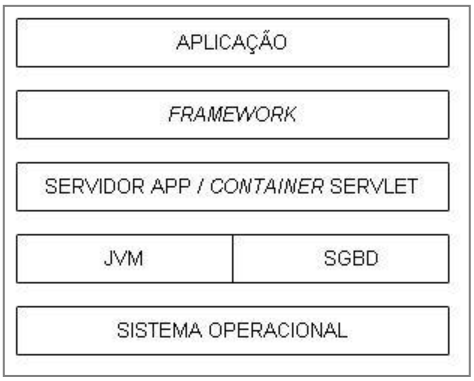


Figura 3 - Arquitetura de uma aplicação *Web* baseada em *Frameworks*
Fonte: (MOTA, 2010)

Segundo Buschmann et al. (1996), Preesmann (1995) e Pinto (2000):

> "um *framework* é definido como um software parcialmente completo projetado para ser instanciado. O *framework* define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. Também são explicitados os lugares ou pontos de extensão (*hot-spots*) nos quais adaptações do código para um funcionamento específico de certos módulos devem ser feitas".

Para Johnson (1991) e Gamma et al (1995),

> "um *framework* é um conjunto de objetos que colaboram com o objetivo de atender a um conjunto de responsabilidades para uma aplicação específica ou um domínio de aplicação".

Já Segundo Mattsson (1996, 2000),

> "um *framework* é uma arquitetura desenvolvida com o objetivo de atingir a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização".

2.2.2 Construção

A construção de um *framework* é feita a partir da observação de um problema ocorrido durante o desenvolvimento de uma aplicação, e que este problema se repete várias vezes e a solução utilizada para resolvê-lo pode futuramente se tornar um *framework*, tornando produtivo o desenvolvimento quando se deparado com esta mesma situação.

A construção de um *framework* envolve três passos: análise das funcionalidades comuns, definição dos *hot-spots* e projeto do *framework.* Segundo Fiorini (2001 apud Souza, 2004, p. 7), o primeiro passo é realizar uma observação das aplicações que já foram ou estão sendo desenvolvidas e que tiveram alguma situação em comum, a partir desta análise verificar quais componentes são reusáveis. Utilizando o conceito de programação orientada a objetos, criam-se classes abstratas para a utilização em alguma aplicação que tiver o mesmo problema. (SOUZA, 2004).

Após realizar a análise, o próximo passo é definir o hot-spots. Depois de saber quais pontos serão fixos, é preciso identificar as situações que variam em todo o contexto da aplicação, e que vão precisar de adaptações para ser utilizada. (SOUZA, 2004).

O ultimo passo para a construção do *framework* é o projeto. Nesta fase, são amarradas as atividades de todos os passos realizados anteriormente, procurando aplicar padrões de projeto para que o produto se torne reusável e cumpra seu objetivo; nesta etapa também são feitas as correções de determinados problemas e remoção de componentes não utilizados, além de modificar alguns componentes para terem novas funcionalidades. Estas observações surgem a partir do uso do *framework* em vários projetos. (SOUZA, 2004).

Segundo Johnson (1992 apud Souza 2004, p.7), um *framework* deve ser uma solução reusável, estável e possuir uma boa documentação. Caso essa documentação não esteja presente, corre-se o risco de o *framework* nunca ser usado. A documentação deve ser feita com o intuito de facilitar a aplicação do *framework* em outros projetos, sendo que alguns aspectos que a documentação deve possuir são:

- Propósito: Define para que serve e quais os problemas o *framework* soluciona;
- Utilização: Descreve como construir uma aplicação utilizando os componentes já prontos;
- Detalhes: Define como os objetos participantes se relacionam.

Depois de documentado o *framework* já está pronto para ser utilizado por desenvolvedores que se depararem com o mesmo problema.

2.2.3 Classificação

Segundo Sauvê (2004 apud Souza 2004), existem três classificações para *frameworks*:

- Middleware: é comumente utilizado em sistemas distribuídos, também provê a integração entre sistemas.
- Suporte: Auxilia no desenvolvimento da infraestrutura da aplicação, não resolvem o problema por completo. São chamados de *frameworks* Horizontais, a Figura 4, demonstra como são esses tipos de *frameworks*, nota-se que existem uma pequena parte disponível, e restante da aplicação deve ser implementada.

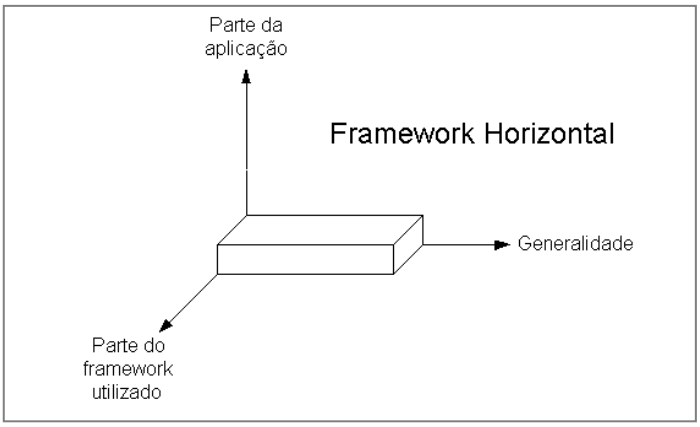


Figura 4 - *Framework* Horizontal
Fonte: (UFCG, 2010)

- Aplicação: É voltado para aplicações, seu objetivo principal é construir aplicações para usuários finais. São chamados de *frameworks* Verticais, a Figura 5 representa a composição deste tipo de *framework*. Nota-se que grande parte da aplicação já está construída, e que apenas uma pequena parte tem que ser implementada, exemplos desses tipos de *frameworks* são: Editores de desenho estruturado, *frameworks* para jogos e monitoração de riscos financeiros.

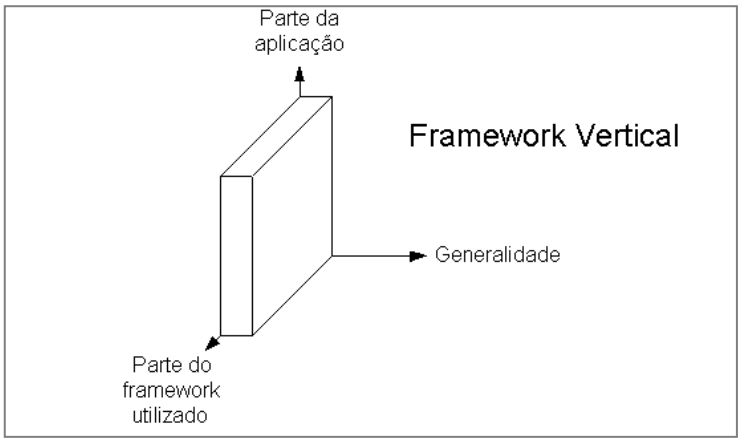


Figura 5 - *Framework* Vertical
Fonte: (UFCG, 2010)

A Figura 4 ilustra a arquitetura dos *frameworks* que serão comparados neste trabalho, como pode ser visto uma pequena parte está pronta para ser utilizado, o restante é preciso ser desenvolvido.

2.2.4 Vantagens e Desvantagens da Utilizações de *Frameworks*

As vantagens da utilização de um *Framework*, de acordo com SOUZA, (2004) são:

- Ganho de tempo no desenvolvimento do projeto, pois já existe uma estrutura semi pronta, e várias funcionalidades já existem;
- Conforme mais pessoas vão utilizando, melhor fica o funcionamento do *Framework*, pelo fato de possíveis problemas já serem sido descobertos e arrumados, além de adicionado novas funcionalidades;
- Acaba sendo necessário implementar o que realmente é preciso, evitando a codificação de toda a aplicação, pois é usado os componentes existentes;
- Como já existe uma estrutura pronta e testada, a probabilidade de erros no código é bem menor.

Por outro lado, a utilização de um *Framework* tem suas desvantagens, de acordo com SOUZA, (2004) são:

- Requer pessoas especializadas: isso acarreta que a equipe faça treinamentos para entendimento do produto, aumentando o prazo para término do projeto;
- A depuração dos programas mais complicada: Se não a disponibilidade de acesso ao código-fonte, fica difícil resolver possíveis erros;
- Mudança no foco do desenvolvimento: Os desenvolvedores passam a ter que entender ideias que não foram proposta pela sua equipe de desenvolvimento;
- Implementação em linguagem especifica: Como são implementados em uma linguagem, perde a portabilidade de poder utilizar outras linguagens em conjunto com o *framework*;

2.3 ESCOLHA DOS *FRAMEWORKS*

Para a realização do estudo comparativo, foram escolhidos os *frameworks* Vaadin e GWT (Google *Web* Toolkit), por possuírem um estilo de programação muito parecido com a programação para *Desktop*, e utilizarem a linguagem Java como elemento principal para o desenvolvimento.

Vaadin é um *framework* de desenvolvimento de aplicações *Web*, que permite aos desenvolvedores criar interfaces de alta qualidade utilizando a linguagem Java. Fornece uma biblioteca pronta para utilizar seus componentes, e também permite a criação de novos

componentes. O objetivo principal é a facilidade de uso, reutilização, extensibilidade, e atender a necessidade de aplicações coorporativas de grandes empresas. (VAADIN, 2012)

O GWT é um conjunto de ferramentas para desenvolvimento *Web* em Java, usado para o desenvolvimento de aplicativos RIA (Rich Internet Application). O objetivo principal do GWT é fazer com que o desenvolvimento se torne mais produtivo, simples, com qualidade, diminuindo a incidência de eventuais erros. O GWT possui um conjunto de componentes visuais rico, o desenvolvedor passa a não se preocupar com Javascript para utilizar Ajax, pois o código feito em Java é transformado em Javascript pelo GWT. (MAGALHÃES, 2009)

2.4 FRAMEWORK VAADIN

O mecanismo central do *framework* Vaadin é a biblioteca Java projetada para o desenvolvimento e manutenção de interface gráfica de alta qualidade. Tendo como ideia principal de programação desenvolver interface *Web* muito parecida com desktop, seu modelo de desenvolvimento se baseia no estilo de programação Swing, SWT e AWT. (VAADIN, 2012)

O Vaadin trabalha com a ideia de modelo de programação ao lado do servidor, ou seja, o gerenciamento da interface do usuário é feita no navegador, e a comunicação Ajax ocorre entre o navegador e o servidor. Como Vaadin trabalha apenas com a linguagem Java, o desenvolvedor não precisa se preocupar com HTML ou Javascript, facilitando a depuração no desenvolvimento da aplicação. (VAADIN, 2012)

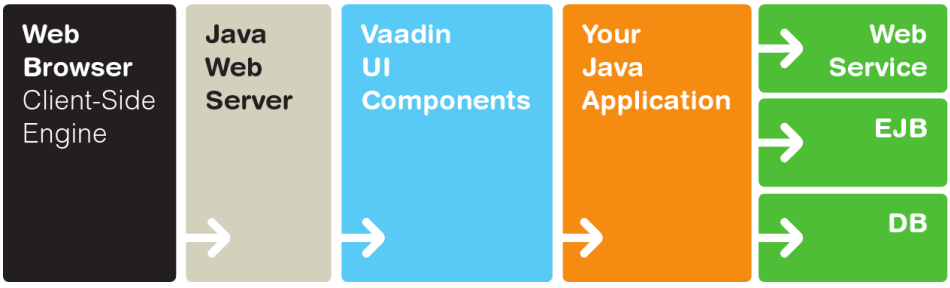


Figura 6 - Arquitetura Geral do Vaadin
Fonte: (VAADIN, 2012)

A Figura 6, ilustra a arquitetura básica da aplicação *Web* Vaadin. Exemplificando, a interface da aplicação no Vaadin é feita no Servidor e enviada para o cliente que é responsável pela criação dos *widget*[4] informados, ou seja, se for preciso mostrar uma lista

[4] Componente utilizado na interface gráfica.

contendo informações de uma tabela em um Combo Box, o Vaadin cria esta lista no servidor e envia ao cliente. (VAADIN, 2012)

Como no lado do cliente é executado Javascript, não há necessidade de o navegador conter *pluguins* para aplicações feitas com o Vaadin. Outro fator importante, é que o Vaadin conta com o apoio do GWT, por utiliza-lo internamente ao lado do cliente, de modo que o desenvolvedor não precisa se preocupar com o suporte ao navegador. (VAADIN, 2012)

Atrás do modelo de desenvolvimento ao lado do servidor, Vaadin faz o melhor uso do conjunto de tecnologias AJAX[5] (Asynchronous Javascript e XML), técnicas que tornam possível a criação de Rich Internet Applications (RIA), que transformam a aplicação interativa como uma feita para desktop. (VAADIN, 2012)

O *Framework* Vaadin faz uso internamente do GWT (Google *Web* Toolkit) como mecanismo de exibição ao lado do cliente. No GWT a aplicação é construída em Java, e compilada em Javascript, liberando a necessidade de o desenvolvedor ter conhecimento nessa linguagem. (VAADIN, 2012) A diferença entre o funcionamento do GWT e Vaadin está na forma de comunicação entre o cliente e servidor. No GWT é feita uma chamada AJAX para a comunicação entre o cliente e o servidor, é criado classes especificas para gerenciar este modelo de comunicação. No Vaadin, também a comunicação AJAX com o cliente, só que a interface do usuário é construída no próprio servidor, ou seja, não existem classes que separam a lógica de programação cliente e servidor.

2.4.1 Arquitetura do Vaadin

Vaadin consiste em uma aplicação *Web* API, contém uma variedade de componentes de interface para o usuário, temas para controlar a aparência, e um modelo de dados que permite que os seus componentes de interface sejam ligados direto com os dados. (VAADIN, 2012).

Uma aplicação que utiliza Vaadin é executada como um servlet em um servidor *Web* Java, atendendo as solicitações HTTP. O adaptador de terminal recebe requisições dos clientes através do servidor *Web* Java *Servlet* [6]API, e interpreta a eventos do usuário para uma

[6] A tecnologia Java Servlet fornece aos desenvolvedores *Web* um mecanismo simples e consistente para estender funcinalidades de um Servidor *Web*. Um servlet pode ser quase considerado um applet que é executado no servidor. (ORACLE, 2010)

sessão particular. As sessões são controladas por *Cookies*[7]. Os eventos estão associados com os componentes de interface e entregues para a aplicação, que os trata como *listeners*[8]. Se ocorrer alteração de algum componente no lado do servidor, o terminal de adaptador comunica o navegador que houve uma modificação, gerando uma resposta, após isso o cliente recebe a resposta e realiza a alteração na página do navegador. (VAADIN, 2012)

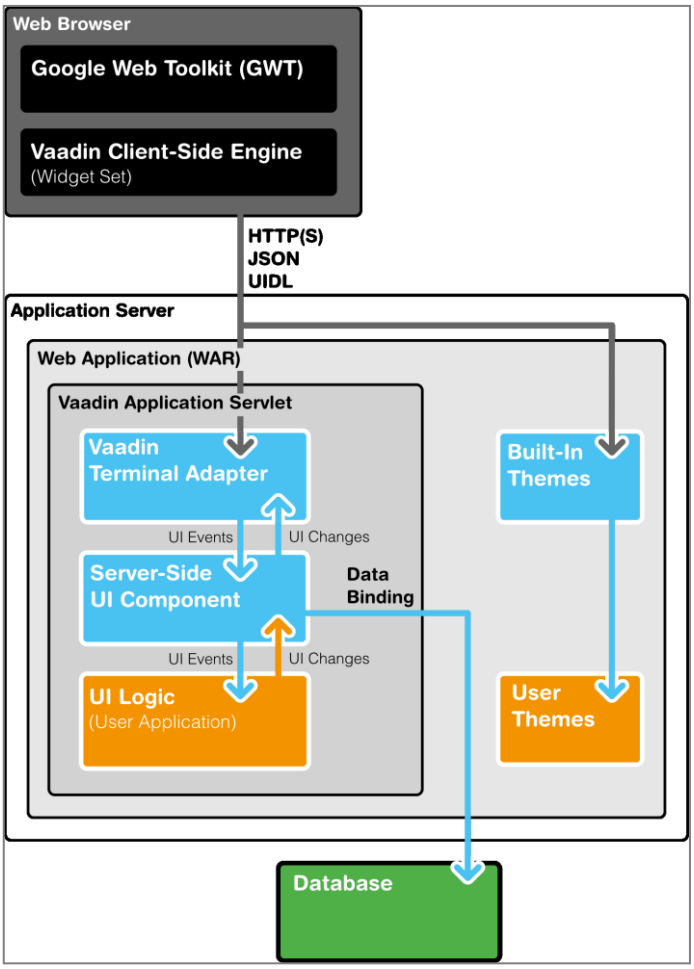


Figura 7 - Arquitetura Vaadin
Fonte: (VAADIN, 2012)

O nível mais alto da aplicação consiste da classe que herda "`com.vaadin.Application`". Ela cria os componentes de interface, recebe os eventos, e faz as alterações nos componentes.

As partes principais da arquitetura e suas funções demonstradas na Figura 7 são:

- User Interface Components: Consiste nos componentes de interface com o usuário criado pela aplicação, na imagem é representado pelo UI Component.

[7] Cookies são uma forma de armazenamento de dados no lado do cliente. (ORACLE, 2010)

- Client-side Engine: Gerencia a renderização do navegador usando Google *Web* Toolkit (GWT). Ele comunica a interação do usuário com a interface, utilizando o adaptador de terminal (Terminal Adaptater) com a User Interface Definition Language (UILD), que é um sistema de comunicação baseado na linguagem JSON. As Comunicações são feitas com solicitações assíncronas HTTP ou HTTPS.
- Terminal Adapter: Os componentes na interface do usuário não se processam automaticamente, precisam de um Terminal Adapter. Essa camada permite aos usuários utilizarem as aplicações vaadin em qualquer navegador *Web*. Quando o usuário realiza alguma ação na página *Web*, esse evento é repassado para o Terminal Adapter como solicitações assíncronas AJAX. O Terminal Adapter devolve os eventos que o usuário realizou para o componente, após isso é atualizada a página com as modificações ocorridas.
- Themes: A interface do usuário faz a separação entre a apresentação e lógica. Enquanto a lógica da interface do usuário é tratada como código Java, a apresentação é definida como temas CSS. Vaadin fornece alguns temas padrões. Pode ser definidas folhas de estilo, modelos HTML que definem layouts personalizados e outros recursos, como imagens.
- UIDL: O Terminal Adapter quando invocado utiliza a linguagem User Interface Definition Language (UIDL). A Comunicação UIDL é feita utilizando JSON (Javascript Object Notation), que é um formato de intercâmbio de dados leve que é especialmente eficaz para comunicação AJAX.
- Events: Toda interação do usuário com a interface, são primeiramente processados no lado do cliente com Javascript, sendo então passado para o Servidor HTTP, Terminal Adapter e para o componente na camada de aplicação.
- Data Model: Utilizando o Data Model os componentes de interface podem ser ligados diretamente com uma fonte de dados, sem a necessidade de um código de controle para atualizar os dados, por exemplo, pode-se vincular um componente Table diretamente com uma resposta SQL.

[8] É um processo separado que é executado no servidor. Ele recebe as requisições do cliente e gerencia o tráfego dessas solicitações no servidor. (ORACLE, 2010)

2.4.2 Criação de um Projeto que Utilize Vaadin

O apêndice A demonstra como instalar e criar um projeto no Vaadin, será exemplificada passo a passo a criação, foi utilizado para este exemplo a IDE Eclipse SDK.

2.4.3 Estrutura do Projeto

Depois de criado o projeto, é necessário efetuar a configuração das bibliotecas a serem utilizadas no desenvolvimento. Assim, deve-se configurar as bibliotecas do Vaadin no diretório `WebContent/WEB-INF/lib`. Para exemplificar a criação do projeto, foi criado um esqueleto da classe de exemplo no diretório `src`, e também foi criado o arquivo `web.xml` já configurado no diretório `WebContent/WEB-INF/Web.xml`. A Figura 8 exemplifica a estrutura do projeto na IDE Eclipse.

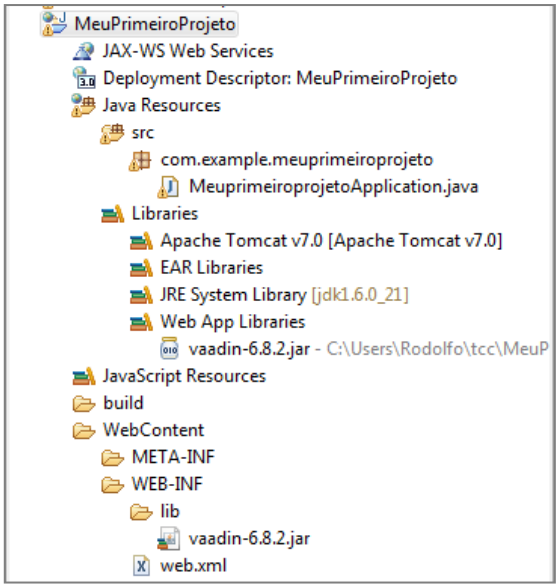


Figura 8 - Estrutura da Aplicação

O código da classe de exemplo da aplicação é apresentado na figura 9.

```
public class MeuprimeiroprojetoApplication extends Application {
    @Override
    public void init() {
        Window mainWindow = new Window("Primeiro Exemplo");
        Label label = new Label("Bem Vindo ao Vaadin");
        mainWindow.addComponent(label);
        setMainWindow(mainWindow);
    }

}
```

Figura 9 - Classe Exemplo

A primeira observação é que toda aplicação Vaadin estende da classe `Application`. E cada instancia desta classe é criada uma sessão com o usuário que invocou, no contexto da aplicação de exemplo, quando o usuário chamar a pagina de “Meu primeiro exemplo”, será chamado o método `init` e executado o que estiver implementado nele.

Quando invocado o método `init` será mostrado o conteúdo da janela principal, que nesse exemplo tem o nome de `mainWindow`, uma aplicação Vaadin pode ter varias janelas, mais sempre será redirecionada para a janela principal.

Após criar a janela, foi criado o componente `Label` com o texto “Bem Vindo ao Vaadin”, e adicionado na janela principal, quando executada a aplicação será mostrada uma página conforme a Figura 10.

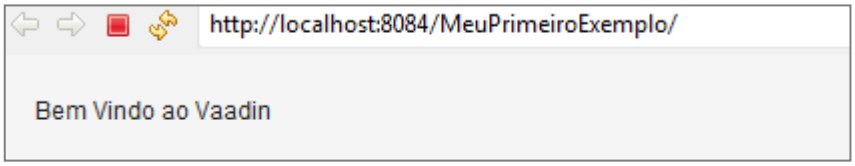


Figura 10 - Aplicação sendo executada

A Figura 11 é o exemplo do arquivo *`web.xml`*, que define o Vaadin *Framework servlet*, a classe de aplicação, e o mapeamento do *servlet*.

```
<?xml version="1.0" encoding="UTF-8"?>
<Web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:Web="http://java.sun.com/xml/ns/javaee/Web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/Web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>MeuPrimeiroProjeto</display-name>
  <context-param>
    <description>Vaadin production mode</description>
      <param-name>productionMode</param-name>
      <param-value>false</param-value>
  </context-param>
  <servlet>
    <servlet-name>Meuprimeiroprojeto Application</servlet-name>
    <servlet-class>com.vaadin.terminal.gwt.server.ApplicationServlet</servlet-class>
    <init-param>
    <description>Vaadin application class to start</description>
      <param-name>application</param-name>
      <param-value>com.example.meuprimeiroprojeto.MeuprimeiroprojetoApplication</param-value>
    </init-param>
  </servlet>
    <servlet-mapping>
      <servlet-name>Meuprimeiroprojeto Application</servlet-name>
      <url-pattern>/*</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
      <welcome-file>index.html</welcome-file>
      <welcome-file>index.htm</welcome-file>
      <welcome-file>index.jsp</welcome-file>
      <welcome-file>default.html</welcome-file>
      <welcome-file>default.htm</welcome-file>
      <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
  </Web-app>
```

Figura 11 - Web.xml Vaadin

O arquivo `Web.xml` é um componente padrão em Java EE que descreve as funcionalidades e características da aplicação *Web* para o servidor. Na figura 11, o código da linha 14 é responsável pelo gerenciamento da implantação da aplicação, pois a classe da aplicação é especificada informando seu nome para o servlet, o servlet é então ligado a uma URL em um caminho padrão para servlets Java.

Portanto, o arquivo Web.xml é um descritor que informa para o servidor como ele deve gerenciar a aplicação *Web*, sendo obrigatório em qualquer aplicação Vaadin

2.5 *FRAMEWORK* GWT (GOOGLE *WEB* TOOLKIT)

O GWT é um *framework* de código aberto, com foco no desenvolvimento de aplicações RIAs para desenvolvedores Java. Estas aplicações são muito parecidas com as desenvolvidas em *desktop*, normalmente são escritas em Javascript. Mas o Javascript não é nem um pouco parecido com a linguagem Java, portanto, seu modelo e pratica de desenvolvimento é diferente. (SMEETS, BONESS e BANKRAS, 2009).

Entretanto, o GWT faz com que seja desenvolvida aplicações Javascript em Java. Isto é realizado pelo compilador de Java para Javascript. Este compilador tem o trabalho de fazer com que todo o código desenvolvido em Java seja traduzido para Javascript, com isso, o GWT consegue lidar com a quase todos os *quirks* [9]dos navegadores, permitindo que o desenvolvedor se concentre apenas com a linguagem Java, e não perdendo tempo codificando para diferentes tipos de navegadores. (SMEETS, BONESS e BANKRAS, 2009).

Segundo (SMEETS, BONESS e BANKRAS, 2009), existem algumas considerações gerais antes de iniciar o desenvolvimento com GWT, são:

- Não é indexado pelo mecanismo de busca: aplicações em Ajax (consequentemente as aplicações em GWT) não são bem indexadas por mecanismos de busca. Pelo fato do mecanismo de busca não aceitar Javascript, só conseguem enxergar a página básica que faz a hospedagem da aplicação Ajax, não conseguem ter uma visão dinâmica dos elementos que dependem do Javascript.
- Não degradam de forma suave: Isso reflete o fato de quando os usuários das aplicações desabilitam o Javascript, eles só conseguem visualizar a página

[9] Quirks: é o modo como os navegadores interpretam o CSS. (PEREIRA, 2005)

básica, se tornando obrigatório habilita-lo para que sua aplicação possa ser utilizada.

- Falta de transparência na separação entre o código e estilo/layout: isso, objetivamente, é o fato dos estilos e aparências estarem sendo utilizado no código Java, ou seja, o GWT não serve para *designers*, pois estes preferem utilizar em suas aplicações HTML puro, com locais de inserção de conteúdos dinâmico.

2.5.1 Arquitetura GWT

Segundo (SMEETS, BONESS e BANKRAS, 2009), o GWT pode ser divido em três partes principais, sendo que a ultima tem várias subdivisões:

- Compilador de Java para Javascript: Essa parte é a mais importante do GWT, é o mecanismo principal de sua existência. Isso faz com que o GWT se torne uma ferramenta poderosa para desenvolvimento de aplicações RIAs. O compilador faz com que todo código desenvolvido em Java seja traduzido para Javascript.
- Biblioteca de emulação JRE: Trabalhar com Javascript é totalmente diferente do processo de desenvolver em Java, portanto, para que o processo de compilação de Java para Javascript funcione, é preciso emular os construtores e classes do núcleo Java, da maneira que isso tudo possa ser traduzido para um código que consiga executar Javascript.
- Biblioteca de geração UI: Essa parte do GWT é dividida em várias partes. Isso é todo o código base fornecido pelo GWT, incluindo os componentes UI, suporte RPC, e gerenciamento do histórico entre outros.

Quanto aos itens, o primeirose refere ao elemento principal do GWT, que é o compilador Javascript, a história sobre o desenvolvimento do GWT, começou quando a Google iniciou o desenvolvimento de aplicações RIA, e acabou por ter alguns problemas ao decorrer do desenvolvimento. Acabaram por fazer uma análise de qual seria a melhor maneira de desenvolver aplicações RIAs e acabaram por decidir em fazer o GWT, sendo que a ideia principal seria que a aplicação fosse desenvolvida com a linguagem Java. O maior desafio foi desenvolver um compilador que conseguisse transformar Java em Javascript, e também que o desenvolvedor não se preocupasse com os diferentes navegadores. No final, acabaram

conseguindo desenvolver o GWT com todos esses requisitos fundamentais para sua utilização. (SMEETS, BONESS e BANKRAS, 2009)

O compilador é um programa Java que pode ser iniciado rodando a classe com.google.gwt.dev.GWTCompiler. Uma das partes fundamentais do compilador, é que ao contrario do compilador Java normal, ele só compilará o código que realmente é utilizado dentro do módulo, evitando gerar para o cliente código Javascript não utilizado. O problema disto é que não é permitido o carregamento das classes em tempo de execução, por não serem compiladas para Javascript, por isso é necessário que o código seja referenciado de alguma forma para que possa ser incluído em tempo de execução. (SMEETS, BONESS e BANKRAS, 2009)

Para finalizar, a Figura 12 é um exemplo do que é gerado pelo compilador quando código Java é traduzido para Javascript.

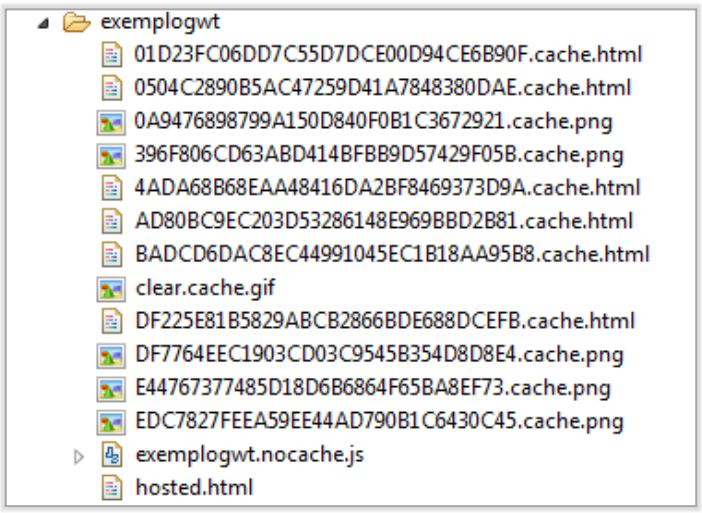


Figura 12 - Saída gerada pelo compilador

O principal fator a se notar é que nesta saída os resultados da compilação são vários arquivos Javascript, que são um para cada plataforma suportada. O `exemplogwt.nocache.js` tem a responsabilidade de carregar o arquivo correto para cada plataforma.

A segunda parte se refere a biblioteca de emulação JRE, que é dividida em duas partes, a primeira lidar com o suporte da linguagem Java para saber o que pode e o que não pode utilizar para a compilação para Javascript, já a segunda parte se refere as classes que são emuladas pelo GWT.

Sobre o suporte a linguagem, pode-se utilizar toda extensão Java para desenvolver aplicações que serão compiladas para Javascript pelo GWT, mais existem algumas diferenças que são importantes no desenvolvimento da aplicação, a baixo a lista dessas observações:

- Suporte a Long: O Javascript não tem representação de Long nas suas especificações de linguagem, portanto não terá como passar um código Long do código Java para os métodos Javascript Native Interface (JSNI).
- Tratamento de exceções: funcionam como em todas aplicações Java, mais existem duas exceções que precisam ser informadas, primeiro, o método `getStackTrace` não retorna nada de útil quando está rodando como Javascript, segundo, as exceções default, como Null Pointer Execept e `OutOfMemoryError` não será incluso no código, ou seja, o desenvolvedor terá de fazer uma verificação de Null por conta própria.
- Thread único: os interceptadores do Javascript são mono-thread, então a aplicação também será. Portanto, os métodos que são relacionado a thread e palavras chaves (exemplo `synchronized`), não terão utilidade nenhuma, e será ignorado pelo compilador GWT.
- Sem reflexão: Não existe a possibilidade de carregar classes dinamicamente por reflexão, pelo fato do compilador GWT ter que saber sobre cada classe e método desenvolvido na aplicação, para poder criar um arquivo Javascript para cada plataforma.

Em relação às classes emuladas, pode-se utilizar quase toda a linguagem Java para desenvolver a aplicação, mais como Javascript trabalha de maneira diferente de Java, existem algumas observações. (SMEETS, BONESS e BANKRAS, 2009)

Para saber quais classes podem ser usadas, basta verificar se as classes utilizadas são emuladas pelo GWT, e mesmo para estas classes emuladas, só pode usar os métodos que também são emulados pelo GWT. Os exemplos das classes que podem ser usadas são as dos pacotes `Java.lang` e `Java.util`, além de algumas do subconjunto `Java.io` e `Java.sql`. (SMEETS, BONESS e BANKRAS, 2009)

Até agora foi mostrado as restrições que o desenvolvedor enfrente utilizando GWT, mais a última parte referente a Biblioteca UI, possui vários subconjuntos, como dito anteriormente, esses subconjuntos facilitam o trabalho, abaixo exemplos dos subconjuntos:

- *Widgets* e Layout: Grande parte no desenvolvimento de uma aplicação RIA é na verdade a criação da interface. No nível mais básico, isso se reflete na

utilização de *widgets* e criando a interface referente a cada um. Portanto, uma grande parte do GWT consiste de componentes e classes relacionadas a layout.

- Comunicação com Servidor: Quase todas as aplicações GWT, precisam de algum tipo de back end para buscar e armazenar dados, e também processar alguma ação. Para isso, o GWT utiliza um mecanismo de RPC para comunicação com o servidor Java.
- Suporte a Testes: Uma das vantagens no desenvolvimento de uma aplicação GWT em relação a aplicações normais do Ajax, é que o código escrito pode ser facilmente testado. Pode ser usado as melhores práticas em relação a testes unitários, e também para alguns casos, o GWT fornece testes com o *framework* de testes JUnit.

2.5.2 Criação de um Projeto que Utilize GWT

O apêndice B demonstra como instalar e criar um projeto com GWT será exemplificado passo a passo a criação, foi utilizado para este exemplo a IDE Eclipse SDK.

2.5.3 Estrutura do Projeto

Uma aplicação desenvolvida em GWT consiste de um conjunto de módulos, que é um pacote GWT constituído por código Java, Javascript, arquivos HTML, imagens, definição de dados e o que é preciso para um aplicativo *Web*. Quando criado a estrutura de diretório GWT no eclipse, ela é baseada na estrutura do módulo GWT que ele implementa. (CHU-CARROL, 2012)

Inicialmente, pode-se analisar a estrutura do projeto criado como exemplo da Figura 13, que se trata de uma aplicação onde o usuário manda uma mensagem para o servidor, e este responde com informação enviada pelo cliente. No Projeto, há um conjunto de bibliotecas GWT, um diretório fonte com o nome de src e também um diretório de destino com o nome de war.

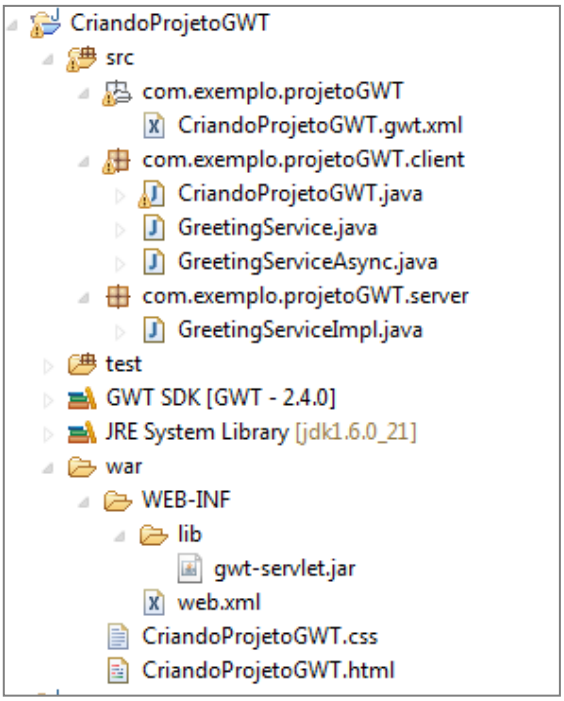


Figura 13 - Estrutura Projeto GWT

O diretório fonte é dividido em três partes: a declaração do módulo, o pacote de código Java ao lado do cliente, e também o pacote Java ao lado do servidor.

O pacote do lado do servidor identificado com o nome "`com.exemplo.projetoGWT.server`", é ilusoriamente simples, pelo fato do GWT gerar a parte interna no servidor.

O cliente é constituído por três arquivos. A `CriandoProjetoGWT` é classe principal da aplicação, as outras duas classes chamada de `GreetingService` e `GreetingServiceImpl` fazem parte da configuração para a chamada de procedimento remoto GWT, essas duas classes definem as declarações necessárias que o GWT precisa para permitir que seja desenvolvido aplicativos cliente/servidor AJAX sem precisar configurar explicitamente o XMLHttpRequests.

A maneira que essas três parte se juntam é configurada na declaração do modulo, que neste exemplo está no pacote com.exemplo.projetoGWT com o nome de `CriandoProjetoGWT.gwt.xml`. A Figura 14 é o exemplo do módulo.

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='criandoprojetogwt'>
  <inherits name='com.google.gwt.user.User'/>
  <inherits name='com.google.gwt.user.theme.clean.Clean'/>
  <entry-point class='com.exemplo.projetoGWT.client.CriandoProjetoGWT'/>

</module>
```

Figura 14 - Módulo GWT

- O elemento `rename` faz a manipulação da URL do GWT, por exemplo o GWT informara o servidor que a URL terminara com o nome `criandoprojetogwt`.
- Os módulos podem herdar propriedades de outros módulos, sendo semelhante com herança em orientação a objetos. O aplicativo é um submódulo de `com.google.gwt.user.User`, que é um módulo padrão para interface de usuário. A grande parte dos componentes, o mecanismo de chamada de procedimento remotos e também a infraestrutura básica dos servlets no servidor, é herdada com está declaração.
- Parte do motivo que o GWT possui herança e também módulos, é fato do GWT poder criar herança com arquivos CSS que definem a aparência dos componentes da aplicação, ou seja, pode-se alterar a aparência do aplicativo herdando de um módulo de estilo diferente.
- Toda aplicação GWT precisa de um ponto inicial que é semelhante a uma função *main*. Neste exemplo, é a classe `CriandoProjetoGwt`. No módulo precisa ser informado qual é o ponto inicial da aplicação.

Em um módulo GWT, a parte de interface do usuário é definida em um arquivo HTML, este arquivo não é considerado código fonte, por isso não é colocado no diretório src, e sim no war. É um recurso estático que terá informações que serão utilizadas pelo código. A Figura 15 é o arquivo HTML deste exemplo.

```
<!doctype html>

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <link type="text/css" rel="stylesheet" href="CriandoProjetoGWT.css">
    <title>Web Application Starter Project</title>
    <script type="text/javascript" language="javascript"
    src="criandoprojetogwt/criandoprojetogwt.nocache.js"></script>
  </head>

  <body>

  </body>
</html>
```

Figura 15 - Arquivo HTML

Como pode ser visto, é um arquivo HTML padrão, começando com a declaração da tag HTML padrão, e também do doctype, além dos blocos de cabeçalho com os rótulos comuns. A tag `<script>` é a parte mais importante do arquivo HTML, é nela que é referenciado o arquivo Javascript gerado pelo compilador, neste caso o arquivo é chamado de `criandoprojetogwt.nocache.js`.

Todo o código Java gerado na classe inicial que pode ser visto da Figura 16, será retornado dento da tag `<body>`, poderia também colocar algum componente com um id especifico e referencia-lo na classe `EntryPoint` Java.

```
public class CriandoProjetoGWT implements EntryPoint {

    @Override
    public void onModuleLoad() {
        // TODO Auto-generated method stub
         final Label lblServerResposta = new Label();
         final TextBox txtMensagem = new TextBox();
         Button btnEnviar = new Button("Enviar para o servidor");
         VerticalPanel verticalPanel = new VerticalPanel();
         verticalPanel.add(new Label("Escreva seu texto: "));
         verticalPanel.add(lblServerResposta);
         verticalPanel.add(txtMensagem);
         verticalPanel.add(btnEnviar);
         RootPanel.get().add(verticalPanel);
         final AsyncCallback<String> callback = new AsyncCallback<String>() {
                public void onSuccess(String result) {
                    lblServerResposta.setText(result);
                }

                public void onFailure(Throwable caught) {
                    lblServerResposta.setText("falha na comunicação");
                }
            };
            btnEnviar.addClickHandler(new ClickHandler(){
                public void onClick(ClickEvent event) {
                    getService().myMethod(txtMensagem.getText(), callback);
                }
            });
        }
     public static GreetingServiceAsync getService() {
        return GWT.create(GreetingService.class);
    }
```

Figura 16 - Classe EntryPoint

- A classe de ponto inicial seria equivalente a uma função *main*, definindo, é como um programa principal em um programa não *GUI*. A função *main* é retratada no método `onModuleLoad`, como o próprio nome reflete, o método é chamado quando o módulo GWT é invocado pelo cliente, é neste método que é criado a interface gráfica da aplicação, e também a configuração de eventos dos componentes.
- A primeira coisa a fazer no método `onModuleLoad` é criar os *widgets* da interface gráfica. A maneira de se trabalha não parece nem um pouco com o desenvolvimento de aplicações *Web* comuns, se identificando com programação estilo Desktop.
- O GWT oferece para o desenvolvedor um contexto GUI que é o conteúdo da página HTML, chamado de `rootpanel`. Neste exemplo, foi acessado o

diretório raiz do arquivo, invocando `RootPanel.get()`, passando o componente `VerticalPanel` populado com um label e um button. Se acontecer da página da aplicação conter elementos e estes estejam identificados por um id, então basta utilizar a chamada com get(id).

Até agora foi mostrado toda a estrutura básica para a criação da interface gráfica em GWT, o que falta demonstrar é como funciona a parte de cliente/servidor em GWT.

O código Ajax utilizado na comunicação cliente/servidor não é escrito explicitamente em GWT, em vez disso é feito uma chamada de procedimento remoto (RPC). (CHU-CARROL, 2012)

O RPC se parece com uma chamada normal de método, mais quando invocada é traduzida pelo GWT em uma solicitação do cliente para o servidor, o valor do retorno do RPC é a resposta do servidor para o cliente. Como em qualquer outra aplicação RPC, existe um lado cliente e outro servidor em GWT. (CHU-CARROL, 2012)

Tradicionalmente, o RPC tenta se parecer com uma chamada ao uma função local, ou seja, se fosse chamar uma função fatorial, por exemplo, seria parecido como uma declaração tradicional de uma função, o mesmo se repete para a sua chamada. (CHU-CARROL, 2012)

Para a aplicação desenvolvida de exemplo, há uma chamada remota, onde o cliente envia uma mensagem para o servidor, e este retorna a mesma como resposta. Inicialmente é criada uma interface síncrona para os métodos do serviço, conforme a Figura 17. De modo um pouco estranho, a interface é escrita no pacote ao lado do cliente, pois o GWT é sempre focado no cliente, e este será o usuário da interface, de maneira que fica localizada no cliente.

```
package com.exemplo.projetoGWT.client;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;


@RemoteServiceRelativePath("greet")
public interface GreetingService extends RemoteService {
    public String myMethod(String s);
}
```

Figura 17 - Chamada Remota

- A anotação `RemoteServiceRelativePath`, se refere ao caminho do serviço em relação a URL raiz, ou seja, se o aplicativo estiver no endereço http://gwt.exemplo.com/CriandoProjetoGWT, então o serviço estará disponível no endereço http://gwt.exemplo.com/CriandoProjetoGWT/greet;

- Toda a interface de serviço deve estender a interface GWT `com.google.gwt.user.client.rpc.RemoteService`, quando estendido, está sendo informado que a interface desenvolvida e para serviços GWT, e assim sendo o GWT sabe que o código Java criado deve ser traduzido para Javascript;
- Uma observação importante é que qualquer parâmetro utilizado em método de serviço deve estender de `java.lang.Serializable` ou também a variante específica de GWT `com.google.gwt.user.client.rpc.IsSerializable`;

Além de criar uma interface síncrona, precisa ser criado uma assíncrona. Para isto basta criar outra interface, que seria uma tradução direta da interface síncrona, os métodos contidos nela devem estar exatamente com o mesmo nome e tipo de retorno, além de adicionado um parâmetro final, que é um `AsynCallback`, a Figura 18 é o exemplo da interface assíncrona.

```
package com.exemplo.projetoGWT.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface GreetingServiceAsync {
     public void myMethod(String s, AsyncCallback<String> callback);
}
```

Figura 18 - Interface Assíncrona

A interface assíncrona não tem ligação alguma com a síncrona, e também não precisa usar nenhuma anotação ou herdar alguma classe especial, ela é usada apenas no cliente, os mecanismos de ligação entre as interfaces são feitas pelo GWT, o único propósito desta é fornecer ao cliente uma interface de chamada.

No lado do servidor em GWT é apenas implementado a interface síncrona feita no cliente, isto é feito estendendo `RemoteServiceServlet`, a Figura 19 é o exemplo da interface RPC no lado do servidor.

```
package com.exemplo.projetoGWT.server;

import com.exemplo.projetoGWT.client.GreetingService;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

@SuppressWarnings("serial")
public class GreetingServiceImpl extends RemoteServiceServlet implements
        GreetingService {
    public String myMethod(String s) {
        return "Servidor resposta: " + s;
    }

}
```

Figura 19 - Interface no Lado do Servidor

A anotação `@SuppressWarnings` existe porque a serialização posteriores da máquina virtual Java usa identificadores para cada versão de classe, por exemplo, se existir alguma classe que implemente de `java.lang.Serializable`, e esta classe não tiver um identificador de versão, na execução do programa o Java retornara um aviso, por isso, está anotação impede que o compilador retorne uma mensagem confusa de advertência. Na parte ao lado do servidor é apenas isto que deve ser implementado.

2.5.4 SmartGWT

Ao longo dos últimos anos bibliotecas de componentes foram surgindo e melhorando significativamente as aplicações *Web*, SmartGWT é uma biblioteca baseada em GWT(Google *Web* Toolkit) que fornece um conjunto amplo e avançado de componentes para interface de usuário, é licenciado pelos termos comercias LGPL, SmartGWT também oferece recursos DataSources para utilizar com seus componentes. DataSource são a base da arquitetura SmartGWT. Um DataSource descreve seu modelo de dados de uma maneira muito semelhante a SQL. No DataSource é definido os campos, seus tipos, validações, mascara para os valores, ente outros.

Além da biblioteca de componente, o SmartGWT oferece também um conjunto de bibliotecas no lado do servidor que é o SmartGWT Enterprise Edition (EE), que são varias ferramentas para construir uma aplicação completa em uma arquitetura *Web* moderna. O SmartGWT Enterprise Edition (EE) pode ser integrado com qualquer aplicação pré-existente. (SMARTCLIENT, 2008)

No desenvolvimento da aplicação foi utilizado a biblioteca de componentes SmartGWT. No estudo será desenvolvido um comparativo entre o modo de desenvolvimento GWT utilizando a biblioteca de componentes SmartGWT.

3 MATERIAL E MÉTODOS

As duas aplicações foram desenvolvidas com a plataforma *IDE Eclipse SDK* na versão *Helios Release 3*. Foi utilizada a versão 6 do *Java Development Kit* (JDK), e como Servidor *Web* foi usado *Apache Tomcat* na sua versão 7.

A criação dos diagramas de casos de uso, diagrama de atores, diagrama de sequência, foram realizados com a ferramenta Astah Community *6.2.1*.

A elaboração do banco de dados das aplicações foi feito com a Ferramenta *Mysql Workbench 5.2*. E o banco de dados utilizado foi *MySql Server 5.0.*

Para o desenvolvimento da aplicação com GWT, foi utilizada a biblioteca SmartGWT 3.1, que possui vários componentes para a construção de interface gráficas.

3.1 DESCRIÇÃO DO PROJETO

O projeto desenvolvido tem o objetivo de realizar uma análise comparativa entre os dois *frameworks.* Consiste em uma aplicação que tem o intuito de controlar o histórico do desenvolvimento de uma criança. Este histórico terá todas as informações que foram obtidas nas consultas realizadas pelo médico. Alguns exemplos do que vai ser armazenado nas consultas são: peso, altura, pressão arterial, batimentos cardíacos.

A aplicação também terá dois gráficos para apresentar as informações da evolução da altura e peso da criança.

3.2 REQUISITOS FUNCIONAIS

O sistema terá os seguintes requisitos funcionais:

- Realizar a manutenção do cadastro de usuários.
- Realizar a manutenção do cadastro de crianças.
- Registrar consultas.
- Gerar gráficos para análise do desenvolvimento das crianças. Este gráfico será um demonstrativo de como a criança está evoluindo, os pontos a serem analisados são: peso e a altura.

3.3 CASOS DE USO

O diagrama de casos de uso da aplicação pode ser visto na Figura 20.

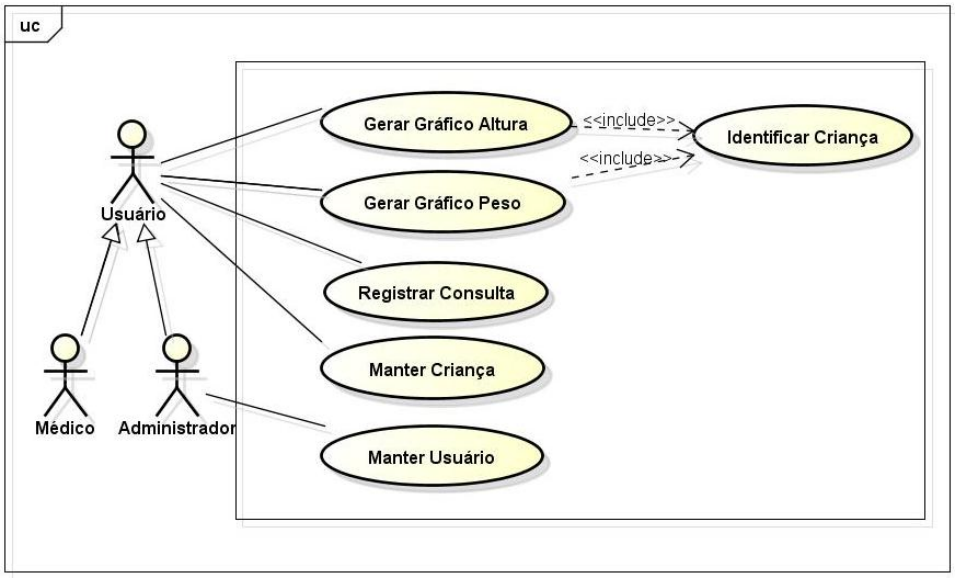


Figura 20 - Diagrama de Casos de Uso

O diagrama de caso de uso do sistema tem dois usuários: Administrador que pode realizar qualquer operação no sistema, e o usuário Médico que só pode gerenciar os cadastros das crianças que é responsável, e também as consultas realizadas por ele com as crianças no qual está vinculado.

Para gerar o gráfico de peso ou altura é necessário identificar qual criança vai ser analisada no gráfico, por isso a existência da ligação *include,* (NOGUEIRA, 2011) define include como: “essa notação é usada para representar sub-fluxos complexos e comuns a vários casos de uso, sempre usados, isto é, necessários”.

3.3.1 DESCRIÇÃO DOS CASOS DE USO

Neste capitulo será exibido às descrições de cada caso de uso representado na Figura 20, sendo possível ter uma melhor análise do funcionamento da aplicação.

Caso de uso: Manter Usuário
Atores: Administrador
Fluxo Principal: 1 O Administrador faz login no *Web*app;

2 Acessa o menu de cadastro de usuários; 3 Caso pesquisar: 3.1 Acessa a aba de pesquisa; 3.2 Informa os parâmetros da pesquisa; 3.3 Clica em pesquisar. 4 Caso incluir: 4.1 Preenche o formulário de cadastro; 4.2 Clica em gravar. 5 Caso editar: 5.1 Pesquisa o usuário (item 3); 5.2 Clica no botão editar; 5.3 Altera os dados necessários; 5.4 Clica em gravar. 6 Caso excluir: 6.1 Pesquisa o usuário (item 3); 6.2 Clica no botão excluir;
Tratamento de exceções: 4 Login já existente. 4.1 O sistema apresenta uma mensagem avisando que o login já está em uso; 4.2 O funcionário informa outro login; 4.3 Volta-se ao fluxo normal. 5 Registro utilizado. 5.1 O sistema apresenta uma mensagem avisando que o registro não pode ser excluído pois está sendo utilizado em outra parte do sistema; 5.2 Encerra-se o fluxo.

Quadro 1 - Manter Usuário

Caso de uso: Manter Criança
Atores: Administrador, Médico
Fluxo Principal: 1 O usuário faz login no *Web*app; 2 Acessa o menu de cadastro de crianças; 3 Caso pesquisar: 3.1 Acessa a aba de pesquisa;

3.2 Informa os parâmetros da pesquisa; 3.3 Clica em pesquisar. 4 Caso incluir: 4.1 Pesquisa o médico responsável e a mãe da criança; 4.2 Preenche o restante das informações; 4.3 Clica em gravar. 5 Caso editar: 5.1 Pesquisa a criança (item 3); 5.2 Clica no botão editar; 5.3 Altera os dados necessários; 5.4 Clica em gravar. 6 Caso excluir: 6.1 Pesquisa a criança (item 3); 6.2 Clica no botão excluir;
Tratamento de exceções: 4 Mãe não cadastrada. 4.1 O usuário registra a mãe da criança; 4.2 Volta-se ao fluxo normal. 6 Registro utilizado. 6.1 O sistema apresenta uma mensagem avisando que o registro não pode ser excluído, pois está sendo utilizado em outra parte do sistema; 6.2 Encerra-se o fluxo.

Quadro 2 - Manter Criança

Caso de uso: Manter Consulta
Atores: Administrador, Médico
Fluxo Principal: 1 O usuário faz login no *Web*app; 2 Acessa o menu de cadastro de consulta; 3 Caso pesquisar: 3.1 Acessa a aba de pesquisa; 3.2 Informa os parâmetros da pesquisa; 3.3 Clica em pesquisar. 4 Caso incluir:

4.1 Pesquisa a criança que vai ser consultada; 4.2 Preenche o restante das informações; 4.3 Clica em gravar. 5 Caso editar: 5.1 Pesquisa a consulta (item 3); 5.2 Clica no botão editar; 5.3 Altera os dados necessários; 5.4 Clica em gravar. 6 Caso excluir: 6.1 Pesquisa a consulta (item 3); 6.2 Clica no botão excluir;
Tratamento de exceções: 5 Criança não cadastrada. 5.1 O usuário registra criança; 5.2 Volta-se ao fluxo normal.

Quadro 3 - Manter Consulta

Caso de uso: Gerar Gráfico Peso
Atores: Administrador, Médico
Pré-condições: Criança identificada
Pós-condições: Gráfico gerado
Fluxo Principal: **1** O usuário faz login no *Web*app; **2** Acessa a aba para gerar gráfico peso; **3** Seleciona a criança a ser analisada; **4** Clica no botão para gerar o gráfico; **5** É aberta uma subpágina com o gráfico gerado;

Quadro 4 - Gerar Gráfico Peso

Caso de uso: Gerar Gráfico Altura
Atores: Administrador, Médico
Pré-condições: Criança identificada

Pós-condições: Gráfico gerado
Fluxo Principal: **6** O usuário faz login no *Web*app; **7** Acessa a aba para gerar gráfico Altura; **8** Seleciona a criança a ser analisada; **9** Clica no botão para gerar o gráfico; **10** É aberta uma subpágina com o gráfico gerado;

Quadro 5 - Gerar Gráfico Altura

3.4 DIAGRAMA DE SEQUÊNCIA

Após analisar a definição dos atores do sistema, os requisitos funcionais e o digrama de casos de uso da Figura 20, é possível se ter uma ideia de como o sistema vai funcionar. O diagrama de sequência deste capítulo vai dividir em três cenários a funcionalidade manter consulta.

UFCG, (2010), define o digrama de sequência como:

> "Consiste em um diagrama que tem o objetivo de mostrar como as mensagens entre os objetos são trocadas no decorrer do tempo para a realização de uma operação".

Os três cenários a serem demonstrados no caso de uso manter consulta são:

- Registrar Consulta: Ilustra os passos a serem executados para realizar o cadastro de uma consulta;

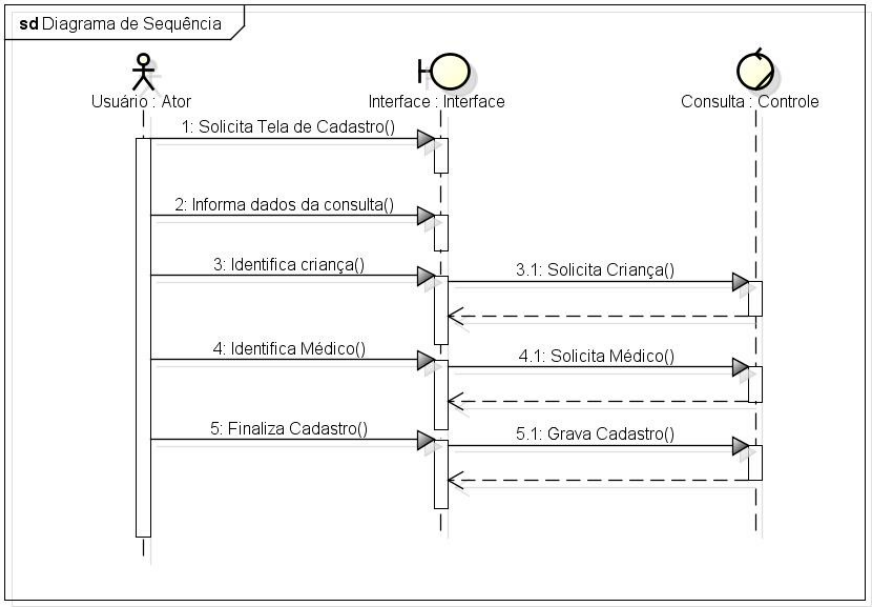


Figura 21 - Digrama de Sequência Cadastrar Consulta

- Editar Consulta: Demonstra como o usuário realiza a edição de uma consulta;

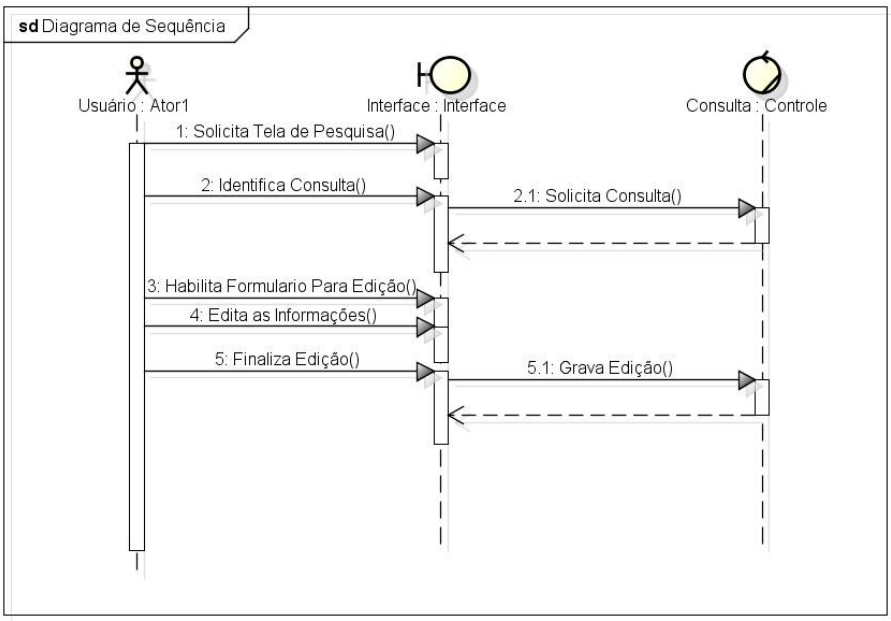


Figura 22 - Diagrama de Sequência Editar Consulta

- Excluir Consulta: Detalha os passos a serem seguidos para excluir uma consulta;

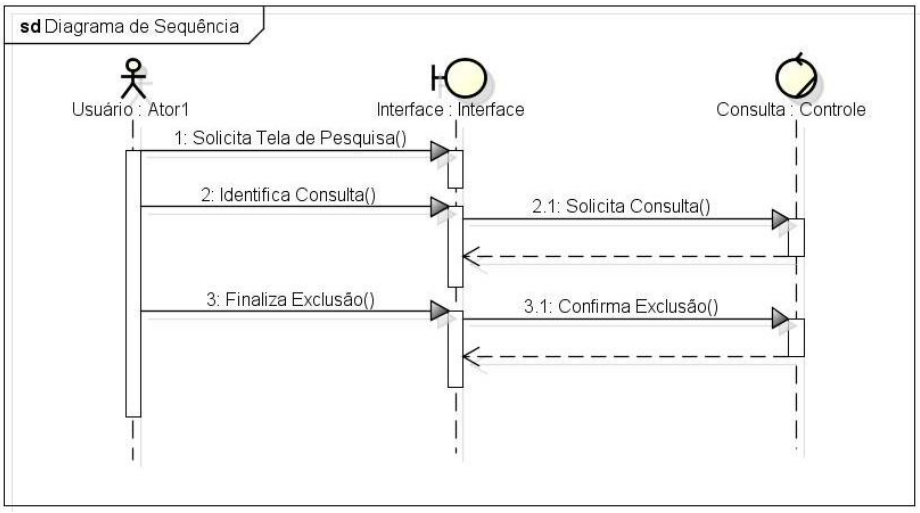


Figura 23 - Diagrama de Sequência Excluir Consulta

3.5 DIAGRAMA DE CLASSES

O diagrama de classes tem o objetivo de demonstrar a estrutura da aplicação, por meio de suas classes e relações. A Figura 24 é o exemplo do diagrama de classe da aplicação

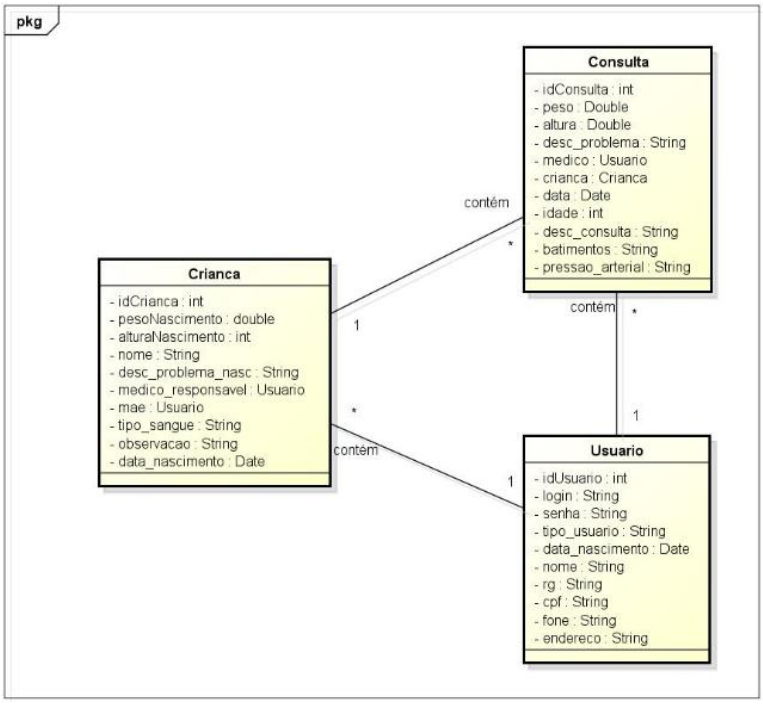


Figura 24 - Diagrama de Classes

3.6 DIAGRAMA DE CONTEÚDO (ÁRVORE DE DADOS)

A Figura 25 retrata o diagrama de conteúdo do sistema na visão do usuário administrador, que tem o objetivo de demonstrar todas as informações que podem ser acessadas na aplicação.

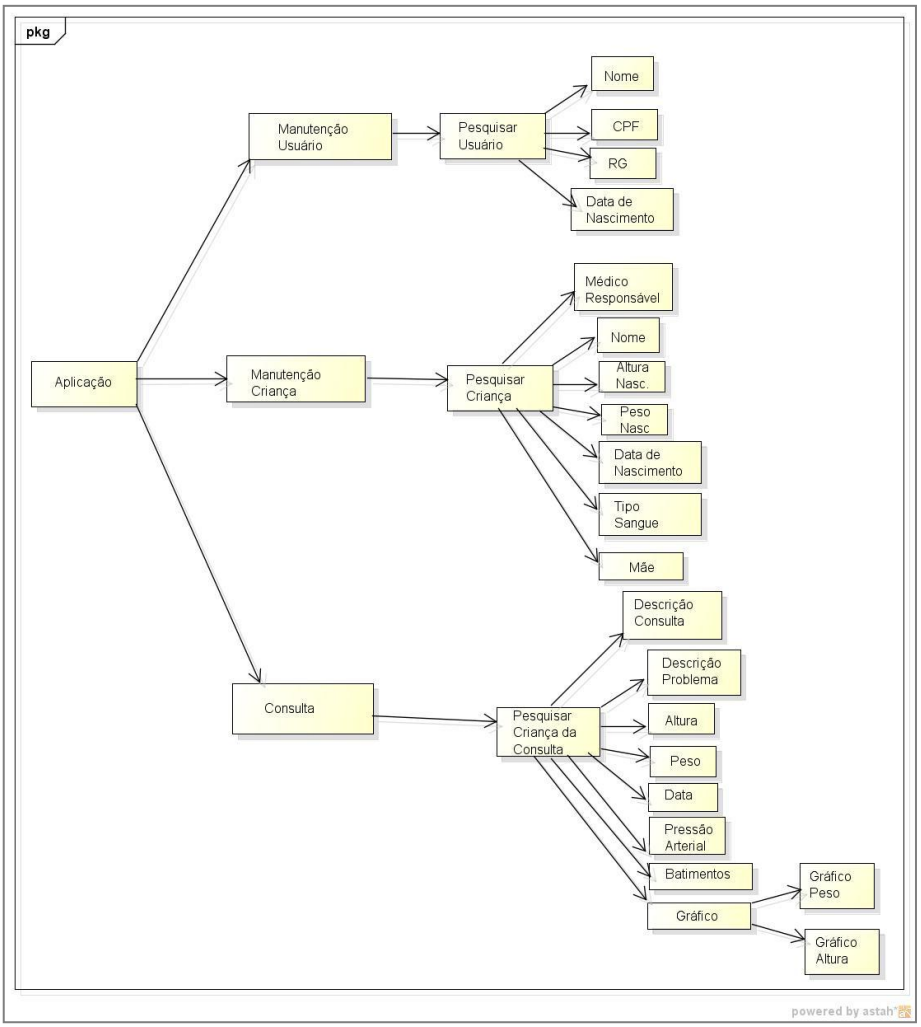


Figura 25 - Diagrama de Conteúdo

3.7 NSU

O NSU (Navigation Semantic Unit) pode ser entendido como um conjunto de informações e estrutura de navegação que auxiliam no entendimento de todo o processo de navegação da aplicação pelos usuários do sistema, no exemplo da figura 26 existem dois usuários no sistema, cada um com determinado nível de acesso, o usuário administrador pode acessar todas as informações na área administrativa da aplicação, já o usuário Médico não pode ter acesso à manutenção usuário, a figura também demonstra como está à estrutura de acesso das paginas da aplicação, informando através das setas o relacionamento entre o acesso.

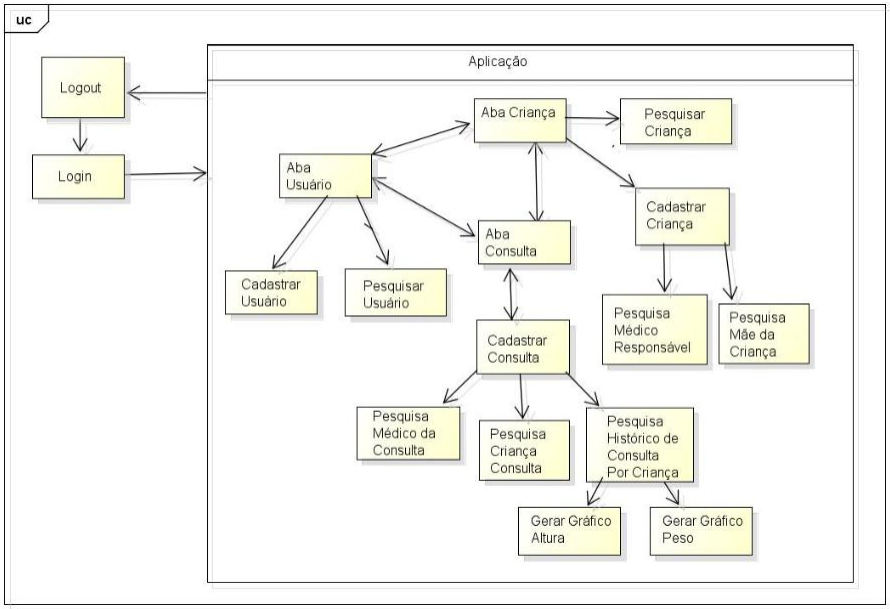


Figura 26 - NSU da aplicação

3.8 WIREFRAME

O Wireframe é um desenho básico que ilustra como será o design da aplicação quando desenvolvida, tem o intuito de demonstrar apenas o essencial. Segundo PINHEIRO, (2011)

> "Os wireframes têm como função simular visualmente a estrutura organizacional das informações de uma interface, sem a preocupação de apresentar uma interface gráfica final".

A Figura 27 retrata o Wireframe da aplicação desenvolvida na visão do usuário administrador:

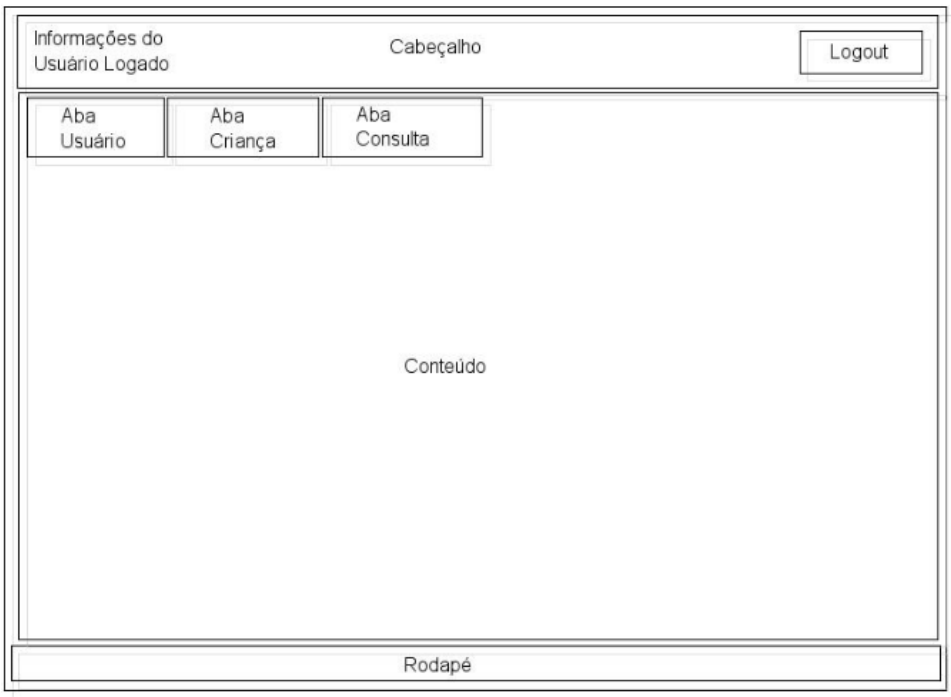


Figura 27 - Wireframe Aplicação

3.9 ESTRUTURA DA APLICAÇÃO

As duas aplicações desenvolvidas tem sua estrutura muito parecidas conforme Figura 28, apresentam praticamente às mesmas classes e estrutura de pacotes, a grande diferença se encontra na maneira como o GWT trabalha. No GWT é criado um pacote client onde contém toda a estrutura de código referente à parte de visualização, exemplos como: formulários, tabelas, validações e entre outros, e o pacote server que contém todas as classes e métodos de persistência da aplicação, é necessário criar classes especificas para comunicação do cliente com o servidor.

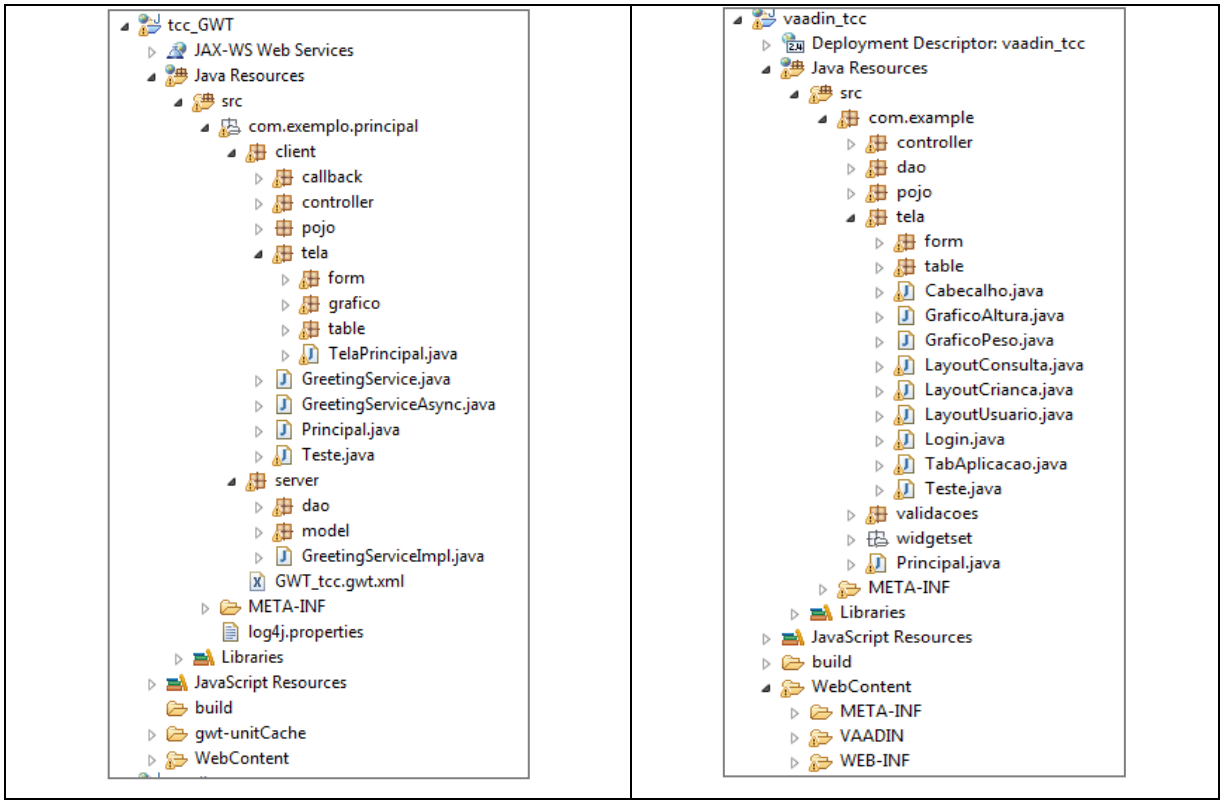


Figura 28 - Estrutura Aplicação GWT ao lado esquerdo, ao lado direito estrutura da aplicação no Vaadin

No Vaadin não é preciso criar classes para a comunicação e ter essa divisão lógica entre cliente e servidor, pois sua programação se baseia ao lado do servidor, ou seja, toda a interface do Vaadin é construída no servidor e enviada para o cliente.

4 RESULTADOS E DISCUSSÕES

Neste capítulo serão analisados os principais fatores no desenvolvimento da aplicação em cada um dos *frameworks* avaliados.

Quando escolhido um *framework* para desenvolver alguma aplicação são levados em consideração vários aspectos, alguns são produtividade e a qualidade que estes *frameworks* levam ao desenvolvedor no decorrer do processo de desenvolvimento, por isso, se torna essencial uma análise desses pontos antes de se iniciar um projeto.

4.1 FATORES ANÁLISADOS PARA O COMPARATIVO

Os fatores analisados para realizar a análise comparativa entre os *frameworks* foram separados em dois aspectos: parte prática e parte teórica.

Na parte prática os pontos levados foram os que causaram maior trabalho e estudo dos *frameworks* durante o desenvolvimento da aplicação, que são:

- Desenvolvimento de Formulários;
- Desenvolvimento de Tabelas;
- Validação de dados;
- Criação de gráficos para a análise de desenvolvimento da criança;
- Diferenças entre o modo de desenvolvimento GWT e Vaadin;

Em relação à parte teórica foram levados em consideração os seguintes requisitos:

- Licença dos *Frameworks;*
- Documentação e Atualizações;
- Quantidade de componentes disponíveis para utilização;

4.2 DESENVOLVIMENTO DO FORMULÁRIO

O exemplo a seguir demonstra como foi desenvolvido o formulário de cadastro de criança utilizado o Framework Vaadin e a biblioteca SmartGWT.

4.2.1 Vaadin

O Vaadin oferece uma maneira muito simples e produtiva para a criação. No Vaadin os campos do formulário podem ser criados a parir dos atributos de uma classe, é necessário encapsular o objeto da classe em um Data Model e atribuí-lo para o formulário, a vantagem disso é que o desenvolvedor não precisa criar todos os campos do formulário, é necessário apenas vincular o objeto da classe no qual quer criar o formulário no adaptador `BeanItem` (Data Model), conforme a Figura 29. Outra vantagem é que o próprio formulário gerencia os campos, não sendo necessário ter o controle de cada um deles.

```
crianca = new Crianca();
criancaItem = new BeanItem<Crianca>(crianca);
```

Figura 29 - BeanItem Classe `Crianca`

A Figura 30 demonstra como foi desenvolvido o formulário de cadastro da classe criança. Alguns aspectos interessantes na criação foram à possibilidade de posicionar os campos no formulário conforme o desejado, atribuir espaçamento entre eles, incluir uma classe responsável por validados os dados de cada campo e também escolher quais campos serão exibidos no formulário.

```
public class FormCrianca extends Form{
    private GridLayout ourLayout;
    public FormCrianca(BeanItem<Crianca> criancaItem) {
        // TODO Auto-generated constructor stub
            setCaption("Cadastro de Criança");
            ourLayout = new GridLayout(4, 4);
            ourLayout.setMargin(true, false, false, true);
            ourLayout.setSpacing(true);
            setLayout(ourLayout);
            setWriteThrough(false);
            setInvalidCommitted(false);
            setFormFieldFactory(new CriancaValidacoes());
            setItemDataSource(criancaItem);
            this.setVisibleItemProperties(Arrays.asList(new String[] {
                    "nome", "dataNascimento", "alturaNasc","pesoNasc" ,
                    "descProblemaNasc","tbUsuario1","tbUsuario2",
                    "tipoSangue","observacao" }));
    }
        @Override
```

Figura 30 - Formulário de cadastro Criança

A Figura 31 ilustra como ficou o formulário de cadastro de crianças da aplicação utilizando o Framework Vaadin.

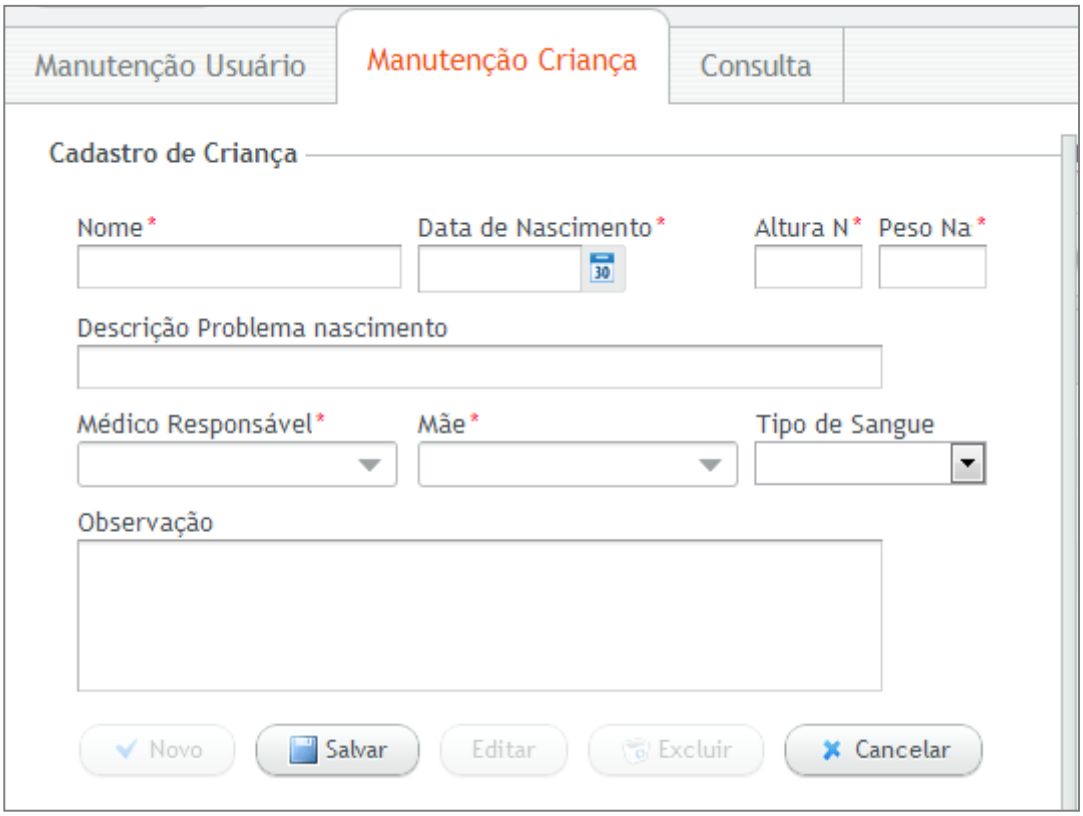


Figura 31 - Formulário Vaadin

4.2.2 SmartGWT

Com a biblioteca SmartGWT a maneira de desenvolver é bem diferente comparando com Vaadin, é preciso utilizar mais código para a criação do formulário. SmartGWT não oferece uma opção para poder vincular a classe com um Data Model e liga-lo com o formulário, é necessário criar todos os campos, existe também uma opção para criar todos os campos e vincula-lo com um DataSource e liga-lo com o formulário, mas neste exemplo não foi utilizado esse método de desenvolvimento, foram criados todos os campos desejáveis e adicionados direto no formulário, a diferença é que agora não é mais o formulário que gerencia os campos da aplicação, é preciso validar diretamente o campo. A Figura 32 ilustra como foi criado uma parte do formulário com SmartGWT.

```
this.setShowResizeBar(true);
this.setWidth("500px");
HeaderItem header = new HeaderItem();
header.setDefaultValue("Cadastro de Criança");
nomeField = new TextItem();
nomeField.setTitle("Nome");
nomeField.setRequired(true);
alturaField = new TextItem();
alturaField.setTitle("Altura Nasc.");
alturaField.setRequired(true);
alturaField.setWidth(50);
alturaField.setKeyPressFilter("[0-9.]");
pesoField = new TextItem();
pesoField.setTitle("Peso Nasc.");
pesoField.setRequired(true);
pesoField.setWidth(50);
pesoField.setKeyPressFilter("[0-9.]");
dateField = new DateItem("date", "Data Nascimento");
dateField.setUseTextField(true);
dateField.setUseMask(true);
dateField.setRequired(true);
dateField.setWidth("150px");
descricaoProbNascField = new TextItem();
descricaoProbNascField.setTitle("Problema Nasc.");
descricaoProbNascField.setRequired(false);
descricaoProbNascField.setWidth(300);
medicoRespField = new TextItem();
medicoRespField.setTitle("Médico Responsável");
medicoRespField.setRequired(true);
PickerIcon searchMedico = new PickerIcon(PickerIcon.SEARCH, new FormItemClickHandler() {
    public void onFormItemClick(FormItemIconClickEvent event) {
         pesquisaMedico();
    }
});
```

Figura 32 - Formulário SmartGWT

Como pode ser analisado, com SmartGWT é utilizado muito mais código, a Figura 32 representa apenas uma parte do formulário criado, faltaram a criação de alguns campos e detalhes da criação do formulário, outra opção que Vaadin oferece em seus *ShowCases* é o componente de *layout* para ajustar os campos no formulário de maneira produtiva, o SmartGWT não oferece nenhum componente produtivo para este tipo de problema.

A biblioteca SmartGWT oferece alguns componentes interessantes para seus campos, que é o caso de poder colocar ícones ao lado dos campos, no formulário desenvolvido foi utilizado um ícone de pesquisa para selecionar o médico responsável pela criança, a Figura 33 mostra como ficou o formulário com a biblioteca SmartGWT.

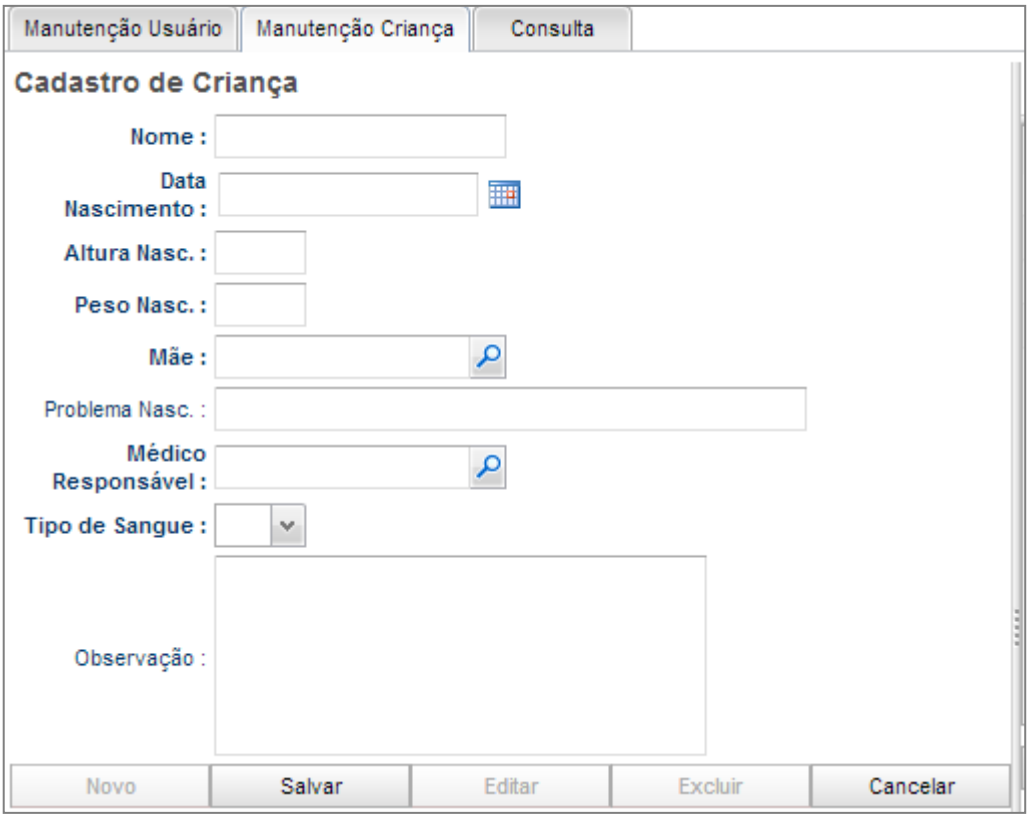


Figura 33 - Formulário SmartGWT

4.3 DESENVOLVIMENTO DE TABELA

Neste exemplo é demonstrado como foi implementado a parte de pesquisa da tela de consulta.

4.3.1 Vaadin

A maneira de se trabalhar com o componente do tipo tabela utilizando o Framework Vaadin se assemelha com o modo como feito o formulário do exemplo anterior. Os dados contidos na tabela são gerenciados por um Data Model, ou seja, a tabela é ligada diretamente com o Data Model. No exemplo da Figura 34 foi criado um container `BeanItemContainer` (Data Model) da classe Consulta, este container contém todos os atributos da classe, que são identificados pelo seus nomes. O container sabe os tipos de cada atributo através dos métodos getters e setters da classe, tornando obrigatória a classe ser publica para conseguir enxergar o tipo dos atributos.

```
public void setBeans(List<Consulta> lista1) {
    beans = new BeanItemContainer<Consulta>(Consulta.class);
    beans.addAll(lista1);
    tabelaConsulta.setContainerDataSource(beans);
    tabelaConsulta.setVisibleColumns(new String[] { "altura", "batimentos", "peso",
            "pressaoArterial","idade" ,"data","tbUsuario","tbCrianca",
            "descConsulta","descProblema" });
    tabelaConsulta.setColumnHeaders(new String[] { "Altura", "Batimentos", "Peso",
            "Pressão","Idade" ,"Data","Médico","Criança","Descrição da consulta",
            "Descrição do problema"});


}
```

Figura 34 - BeanItemContainer Vaadin

Uma das vantagens de utilizar a classe `BeanItemContainer` para gerenciar os campos é que ele aceita receber uma lista com os valores da consulta pesquisada, tornando sua utilização muito produtiva. Outro fator importante é que no componente da tabela aceita que os valores de seus campos sejam formatados através de seu tipo, conforme Figura 35.

```
        protected String formatPropertyValue(Object rowId, Object colId,
                com.vaadin.data.Property property) {
            Object v = property.getValue();
            if (v instanceof Date) {
                Date dateValue = (Date) v;
                return new SimpleDateFormat("dd/MM/yyyy").format(dateValue);
            }
            return super.formatPropertyValue(rowId, colId, property);
        }

    };
```

Figura 35 - Formatando Campo Data Vaadin

A Figura 36 ilustra como foi desenvolvida a tabela com o *framework* Vaadin. Uma observação importante é a possibilidade de escolher quais campos serão exibidos, informando a descrições de suas respectivas colunas.

```
tabelaConsulta = new Table();
tabelaConsulta.setSelectable(true);
tabelaConsulta.setImmediate(true);
tabelaConsulta.setWidth("100%");
tabelaConsulta.setHeight("300px");
beans = new BeanItemContainer<Consulta>(Consulta.class);
beans.addAll(new ArrayList<Consulta>());
tabelaConsulta.setContainerDataSource(beans);
tabelaConsulta.setColumnReorderingAllowed(true);
tabelaConsulta.setColumnCollapsingAllowed(true);
tabelaConsulta.setVisibleColumns(new String[] { "altura", "batimentos", "peso","pressaoArterial","idade" ,
        "data","tbUsuario","tbCrianca","descConsulta","descProblema" });
tabelaConsulta.setColumnHeaders(new String[] { "Altura", "Batimentos", "Peso","Pressão","Idade" ,
        "Data","Médico","Criança","Descrição da consulta","Descrição do problema"});
tabelaConsulta.addListener(new Table.ValueChangeListener() {
      public void valueChange(
            com.vaadin.data.Property.ValueChangeEvent event) {
        consultaSelecionado = (Consulta) event.getProperty().getValue();
      }
});
this.setSpacing(true);
this.addComponent(tabelaConsulta);
```

Figura 36 - Tabela de Consulta

A Figura 37 retrata como ficou a tabela desenvolvida com o *framework* Vaadin:

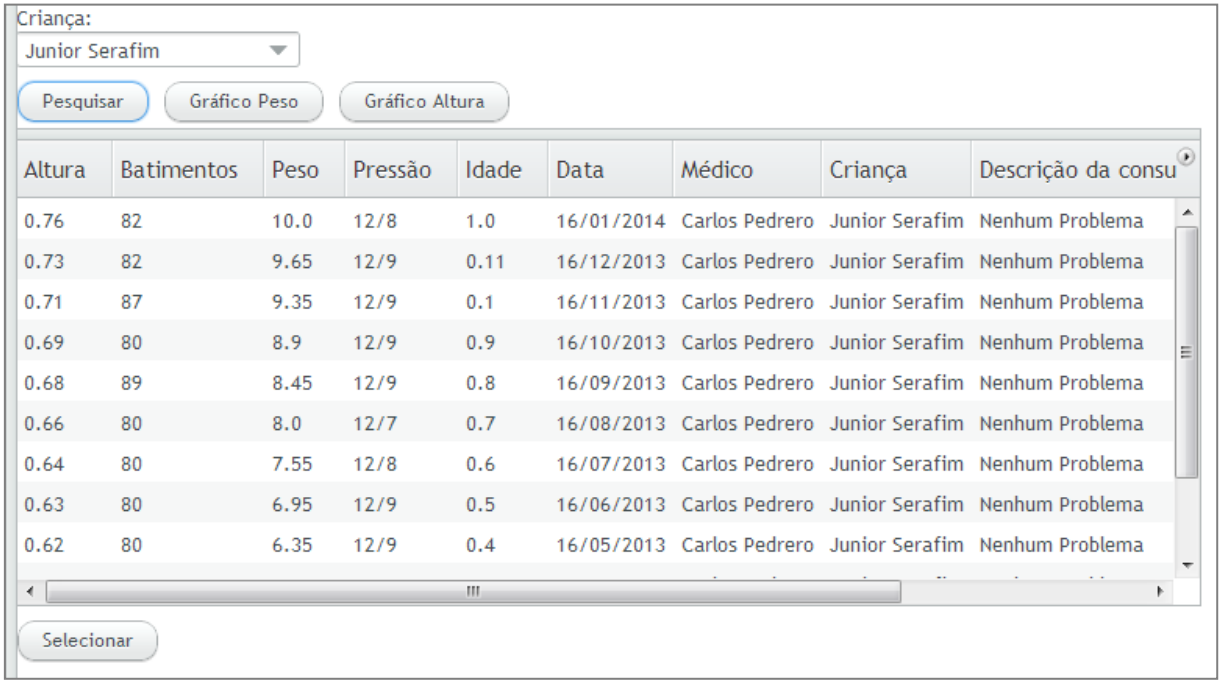


Figura 37 - Exemplo Tabela Vaadin

4.3.2 SmartGWT

A maneira de criar tabela com a biblioteca SmartGWT é muito diferente comparando com o Framework Vaadin. Com SmartGWT não existe uma opção para criar um container com as propriedade de uma classe e liga-lo com a tabela. No SmartGWT é obrigatório criar campo a campo com suas características, gerando maior esforço e menos produtividade na criação da tabela.

Existem varias maneiras de desenvolver tabelas com SmartGWT, o modo usado foi utilizando a classe `ListGridRecord` que é um objeto contendo valores para cada `ListGridField` que é adicionado na tabela, a Figura 38 ilustra a classe `ListGridRecord` utilizada.

```
public class ListaConsultas extends ListGridRecord{

    public ListaConsultas(Integer idConsulta, Double altura, String batimentos,
            Date data, String descConsulta, String descProblema, Double idade,
            Double peso, String pressaoArterial, Usuario tbUsuario,
            Crianca tbCrianca) {
        // TODO Auto-generated constructor stub
        setIdConsulta(idConsulta);
        setAltura(altura);
        setData(data);
        setDescConsulta(descConsulta);
        setIdade(idade);
        setDescProblema(descProblema);
        setPeso(peso);
        setTbUsuario(tbUsuario);
        setTbCrianca(tbCrianca);
        setPressaoArterial(pressaoArterial);
        setBatimentos(batimentos);
    }
```

Figura 38 - Classe ListGridRecord

O objeto `ListGridField` é o que vai ser adicionado como coluna na tabela, obrigatoriamente tem que ser identificado com o mesmo nome de propriedade do objeto da classe `ListGridRecord` para ser identificado e exibido os valores na tabela, a Figura 39 retrata como foi criada a tabela de consulta com SmartGWT.

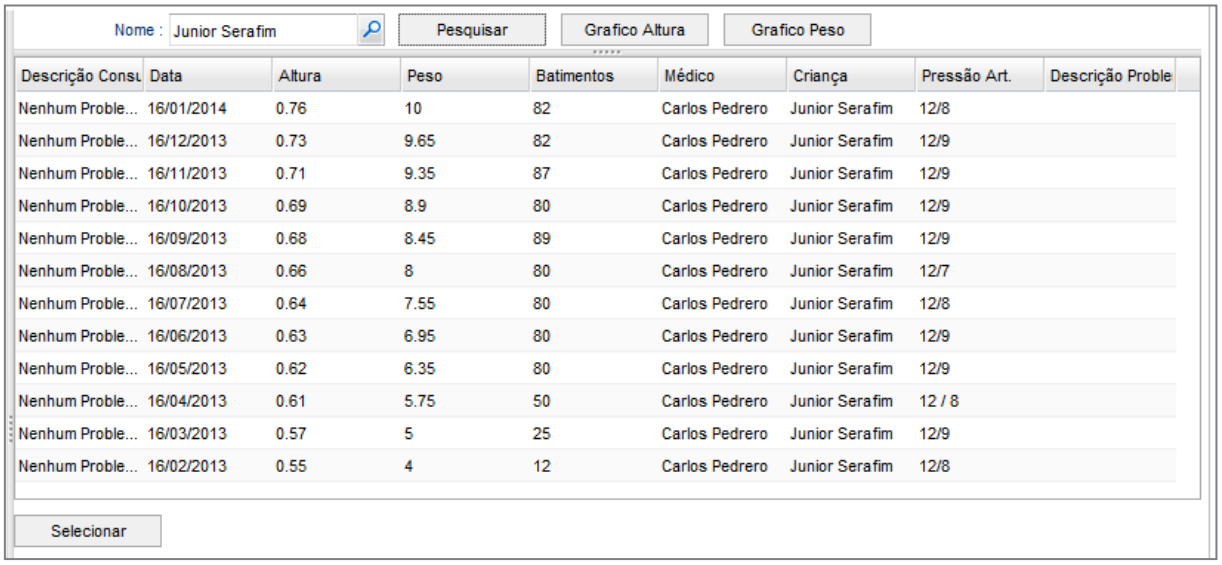
```
public void montaCamposTable(){
    this.setWidth100();
    this.setHeight(300);
    this.addRecordClickHandler(this);
    this.setShowAllRecords(true);
    ListGridField descConsultaField = new ListGridField("descConsulta", "Descrição Consulta");
    ListGridField dataField = new ListGridField("data", "Data");
    ListGridField alturaField = new ListGridField("altura", "Altura");
    ListGridField pesoField = new ListGridField("peso", "Peso");
    ListGridField batimentosField = new ListGridField("batimentos", "Batimentos");
    ListGridField pressaoField = new ListGridField("pressaoArterial", "Pressão Art.");
    ListGridField obsField = new ListGridField("descProblema", "Descrição Problema");
    ListGridField medicoField = new ListGridField("tbUsuario", "Médico");
    ListGridField criancaField = new ListGridField("tbCrianca", "Criança");
    this.setFields( descConsultaField, dataField, alturaField,pesoField,batimentosField,
            medicoField,criancaField,pressaoField,obsField);
    this.setCanResizeFields(true);
}
```

Figura 39 - Tabela de Consulta SmartGWT

Um mecanismo interessante que a biblioteca SmartGWT oferece para os componentes grid em seus *ShowCases* é o de filtro automático das informações contidas na tabela, no qual

não foi utilizado, pois a pesquisa era realizada diretamente no banco de dados não nas informações que tinha na tabela.

A Figura 40 ilustra como ficou a tabela desenvolvida com SmartGWT.



Descrição Consu	Data	Altura	Peso	Batimentos	Médico	Criança	Pressão Art.	Descrição Proble
Nenhum Proble...	16/01/2014	0.76	10	82	Carlos Pedrero	Junior Serafim	12/8	
Nenhum Proble...	16/12/2013	0.73	9.65	82	Carlos Pedrero	Junior Serafim	12/9	
Nenhum Proble...	16/11/2013	0.71	9.35	87	Carlos Pedrero	Junior Serafim	12/9	
Nenhum Proble...	16/10/2013	0.69	8.9	80	Carlos Pedrero	Junior Serafim	12/9	
Nenhum Proble...	16/09/2013	0.68	8.45	89	Carlos Pedrero	Junior Serafim	12/9	
Nenhum Proble...	16/08/2013	0.66	8	80	Carlos Pedrero	Junior Serafim	12/7	
Nenhum Proble...	16/07/2013	0.64	7.55	80	Carlos Pedrero	Junior Serafim	12/8	
Nenhum Proble...	16/06/2013	0.63	6.95	80	Carlos Pedrero	Junior Serafim	12/9	
Nenhum Proble...	16/05/2013	0.62	6.35	80	Carlos Pedrero	Junior Serafim	12/9	
Nenhum Proble...	16/04/2013	0.61	5.75	50	Carlos Pedrero	Junior Serafim	12 / 8	
Nenhum Proble...	16/03/2013	0.57	5	25	Carlos Pedrero	Junior Serafim	12/9	
Nenhum Proble...	16/02/2013	0.55	4	12	Carlos Pedrero	Junior Serafim	12/8	

Figura 40 - Tabela SmartGWT

4.4 VALIDAÇÃO DE DADOS

Neste exemplo será mostrado como foi desenvolvido a parte de validação dos dados no cadastro de usuário.

4.4.1 Vaadin

O Framework Vaadin disponibiliza uma classe para customizar e validar todos os campos do formulário. A Classe se chama `DefaultFieldFactory`, através da propriedade do campo na qual pertence o formulário é adicionado a validação desejada e customizado conforme o necessário, por exemplo, é possível criar campos TextField, ComboBox, DateFiel, PasswordField, entre outros. A classe é ligada ao formulário conforme a Figura 41.

```
setFormFieldFactory(new UsuarioValidacoes());
```

Figura 41 - Validação Usuário

A classe "`UsuarioValidacoes`" da Figura 41 estende a classe `DefaultFieldFactory`. As validações e customizações dos campos é realizado na classe

"`UsuarioValidacoes`", a Figura 42 retrata a customização e validação de alguns campos do formulário.

```
public Field createField(Item item, Object propertyId,
        Component uiContext) {
System.out.println("teste " + item + " - " + propertyId + " - " + uiContext);
    Field f;
   if ("senha".equals(propertyId)) {
        f = createPasswordField(propertyId);
    }else if("tipoUsuario".equals(propertyId)) {
        return tipoUsuario;
    }else if("dataNascimento".equals(propertyId)) {
        f = createDataField(propertyId);
    }else if("fone".equals(propertyId)) {
        f = createMaskara(propertyId);
    }
   else {
        f = super.createField(item, propertyId, uiContext);
   }
    if ("nome".equals(propertyId)) {
        TextField tf = (TextField) f;

        tf.setCaption("Nome");
        tf.setRequired(true);
        tf.setRequiredError("Nome é obrigatório");
        tf.setWidth("13em");
        tf.addValidator(new StringLengthValidator(
                "Nome precisa de 3-25 characters", 3, 25, false));
    } else if ("cpf".equals(propertyId)) {
```

Figura 42 - Validando e Customizando Campos Vaadin

O *framework* Vaadin disponibiliza vários tipos de validação, alguns exemplos são: campo obrigatório, validação no tamanho do campo, máscara e permitir apenas número inteiro, entre outros. Um dos benefícios é poder colocar mensagem para as validações tornando intuitivo para o usuário, a Figura 43 é exemplo do formulário de cadastro de Usuário sendo validado.

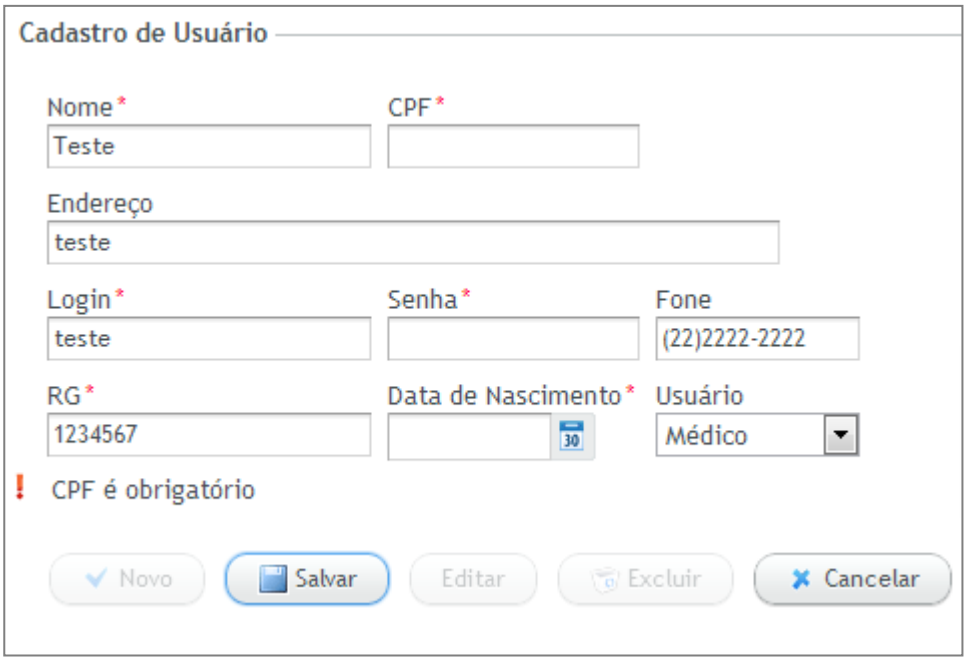


Figura 43 - Validação Cadastro de Usuário

4.4.2 SmartGWT

Com a biblioteca SmartGWT a maneira de realizar a customização e validação das informações é diferente. Existem varias maneiras de validar itens com a biblioteca, isso depende de como foi criado o formulário, no caso deste exemplo a validação foi colocada a cada campo criado e adicionado à customização desejada, conforme o exemplo da Figura 44, a biblioteca oferece uma maneira muito produtiva para validar alguns tipos de campo, um exemplo é permitir apenas o usuário a digitar números em campos numéricos, não permitindo outro tipo de informação.

```
public void criaCampos(){
    nomeField = new TextItem();
    nomeField.setTitle("Nome");
    nomeField.setRequired(true);
    nomeField.setWidth(200);
    cpfField = new TextItem();
    cpfField.setTitle("CPF");
    cpfField.setRequired(true);
    cpfField.setWidth(120);
    rgField = new TextItem();
    rgField.setTitle("RG");
    rgField.setRequired(true);
    rgField.setWidth(120);
    phoneNumberField = new TextItem("fone", "Fone");
    phoneNumberField.setMask("(###) ###-####");
    phoneNumberField.setHint("<nobr>(###) ###-####</nobr>");
    phoneNumberField.setWidth(100);
    enderecoField = new TextItem();
    enderecoField.setTitle("Endereço");
    enderecoField.setRequired(false);
    enderecoField.setWidth(300);
    loginField = new TextItem();
    loginField.setTitle("Login");
    loginField.setRequired(true);
    passwordItem = new PasswordItem();
    passwordItem.setTitle("Senha");
    passwordItem.setRequired(true);
    dateField = new DateItem("date", "Data Nascimento");
    dateField.setUseTextField(true);
    dateField.setUseMask(true);
    dateField.setRequired(true);
    dateField.setWidth("150px");
```

Figura 44 - Validação SmartGWT

Um ponto negativo da biblioteca são os componentes utilizados para validar dados, por exemplo, com o Framework Vaadin é possível colocar mensagens intuitivas e retornar mensagens ao lado do campo validado conforme a Figura 43, a biblioteca SmartGWT não possui este tipo de validação, a sua maneira de validar é retornando apenas um ícone ao lado do campo, conforme Figura 45.

Figura 45 - Validando Formulário SmartGWT

4.5 CRIAÇÃO DE GRÁFICO

Neste exemplo será abordado como foi desenvolvido o gráfico de desenvolvimento da altura da criança.

4.5.1 Vaadin

O *Framework* Vaadin disponibiliza uma biblioteca muito produtiva para o desenvolvimento de gráfico. A biblioteca é chamada de *DChart* e está disponível para a versão 6.8 em diante, está biblioteca é baseada no *plugin* jqPlot.

O *plugin* jqPlot é licenciado pela GPL e MIT, este *plugin* disponibiliza vários tipo de gráfico com *layout* extremamente intuitivo, além de ser de fácil entendimento para o desenvolvimento.

No exemplo desenvolvido com o *Framework* Vaadin, foi criado um gráfico de linhas, a biblioteca oferece vários recursos e modelos interessantes para a criação de gráficos, basta analisar e escolher qual se encaixa melhor para o uso. O interessante da biblioteca é os recursos que ela oferece, um deles é o efeito visual no tracejado informando em qual valor aquele ponto se encaixa, e também uma linha no gráfico que demonstra a média de desenvolvimento no geral, facilitando a análise do médico, demonstrando como a criança está evoluindo perante determinado período comparando com a média, conforme Figura 46.

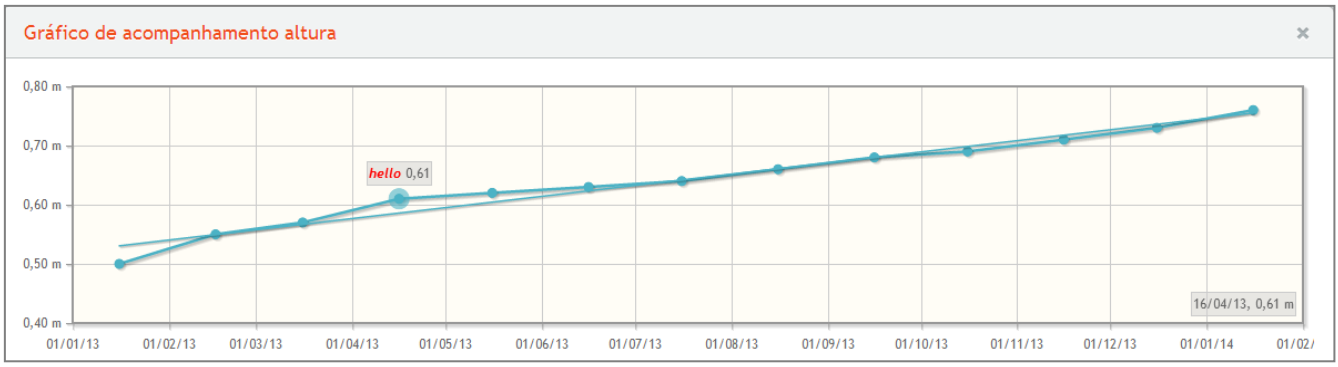


Figura 46 - Gráfico Altura Vaadin

Outro ponto interessante na biblioteca é o fato de seus recursos de efeito da Figura 56 ser de fácil entendimento ao desenvolvedor, além de permitir que os números e datas sejam formatados conforme o desejado, a Figura 47 ilustra como foram colocados os efeitos e formatados os valores do gráfico.

```
47          SeriesDefaults seriesDefaults = new SeriesDefaults()
48           .setTrendline(
49               new Trendline()
50                   .setShow(true));
51
52           Axes axes = new Axes()
53           .addAxis(
54               new XYaxis()
55                   .setRenderer(AxisRenderers.DATE)
56                   .setTickOptions(
57                       new AxisTickRenderer()
58                           .setFormatString("%d/%m/%y"))
59                           .setNumberTicks(consultas.size()))
60           .addAxis(
61               new XYaxis(XYaxes.Y)
62                   .setTickOptions(
63                       new AxisTickRenderer().setFormatString("%.2f m")));
64
65           Highlighter highlighter = new Highlighter()
66           .setShow(true)
67           .setSizeAdjust(10)
68           .setTooltipLocation(TooltipLocations.NORTH)
69           .setTooltipAxes(TooltipAxes.Y)
70           .setTooltipFormatString("<b><i><span style='color:red;'>hello</span></i></b> %.2f")
71           .setUseAxesFormatters(false);
72           Cursor cursor = new Cursor()
73           .setShow(true);
```

Figura 47 - Exemplo Gráfico Vaadin

4.5.2 SmartGWT

A biblioteca SmartGWT também oferece recursos para o desenvolvimento de gráficos. Comparando com o *framework* Vaadin oferece menos recursos visuais para se trabalhar, por exemplo, a biblioteca SmartGWT não possui nenhum recurso de efeito para melhor

visualização, e também menos diversidade em seus *ShowCases*, mais seus componentes são amigáveis para o usuário, e fácil implementação para o desenvolvedor, em poucas linhas de código é possível se desenvolver um gráfico com a biblioteca, a Figura 48 demonstra como foi criado o gráfico com SmartGWT.

```
30                  this.setData(listaConsulta);
31                  this.setFacets(new Facet("periodo", "Data"));
32                  this.setValueProperty("altura");
33                  this.setChartType(ChartType.LINE);
34                  this.setTitle("Desenvolvimento por período");
35                  this.setValueTitle("Altura");
36                  this.setHeight100();
37                  this.setWidth100();
```

Figura 48 – Exemplo de Código SmartGWT

O exemplo de como ficou o gráfico na aplicação Web pode ser visto na Figura 49.

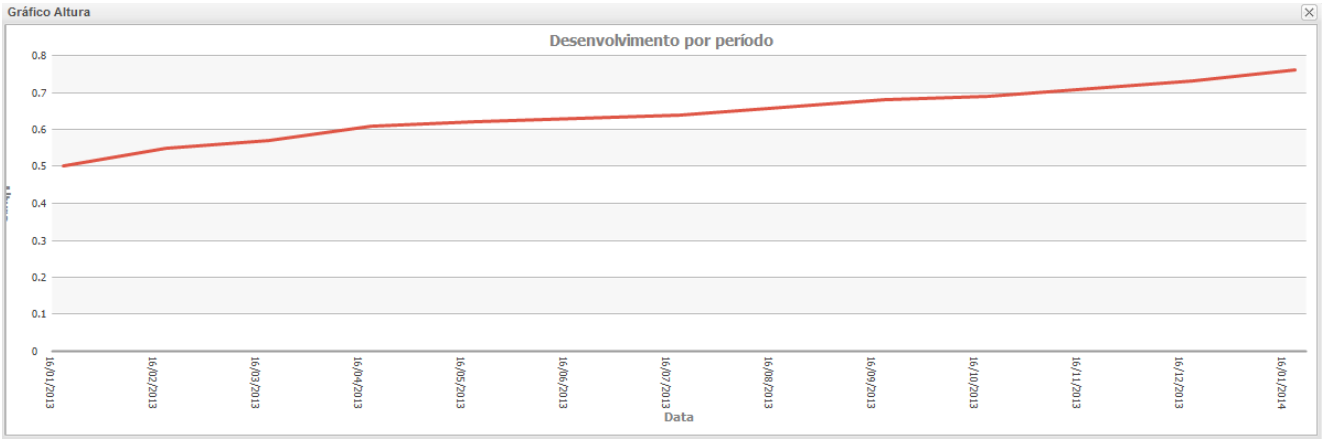


Figura 49 - Gráfico Altura SmartGWT

4.6 DIFERENÇAS ENTRE VAADIN E GWT

Vaadin e GWT têm uma arquitetura de funcionamento diferente, a programação no Vaadin é voltada para o servidor, por exemplo, quando criado uma tela com o *Framework*, está tela é montada no servidor e enviada para o cliente que exibe a mesma, existe um mecanismo no Vaadin para que a interação entre cliente e servidor seja em AJAX, tornando a interação assíncrona e intuitiva para o usuário.

Esta maneira de programação do Vaadin torna o *framework* muito produtivo comparando com GWT, pelo fato de não se preocupar com a criação de classes para a comunicação cliente e servidor, pois com Vaadin tudo é feito no servidor. A classe que estende de `Application` é o que da inicio a aplicação com o *framework* Vaadin, após criar

esta classe já é possível iniciar o desenvolvimento de toda a aplicação no método `init`, sem se preocupar com outra linguagem além de Java, tornado o desenvolvimento muito parecido com *Desktop*, a Figura 50 é o exemplo da classe `Application` da aplicação no Vaadin.

```
11 public class Principal extends Application  implements ApplicationContext.TransactionListener{
12     TabAplicacao tab;
13     Window telaSistema;
14     Usuario usuarioSessao;
15        private static ThreadLocal<Principal> currentApplication =
16            new ThreadLocal<Principal> ();
17     @Override
18     public void init() {
19         setTheme("mytheme");
20         getContext().addTransactionListener(this);
21         setMainWindow(new Login());
22     }
```

Figura 50 - Classe Application Vaadin

O GWT tem uma arquitetura voltada para o cliente, toda a estrutura de código é desenvolvida no cliente e compilada em *Javascript*, o GWT abstrai o possível a programação no lado do servidor. Para a comunicação entre cliente e servidor, o GWT utiliza RPC, que traduzindo significa chamada de procedimento remoto, ou seja, o cliente faz uma requisição para um serviço situado em outro endereço.

A maneira que o GWT trabalha com RPC é vista do seguinte modo:

- Primeiramente é declarada a interface síncrona no cliente com os métodos que devem ser implementados no servidor, a interface `GreetingService` da Figura 51 é o exemplo na aplicação;

```
14 @RemoteServiceRelativePath("greet")
15 public interface GreetingService extends RemoteService {
16
17     public Usuario gravaUsuario(Usuario u) throws Exception;
18     public List<Usuario> listaUsuarios(String usuario) throws Exception;
19     public List<Usuario> listaUsuarios() throws Exception;
20     public List<Usuario> buscaUsuariosMedico() throws Exception;
21     public List<Usuario> buscaUsuariosMae() throws Exception;
22     public void excluiUsuario(Usuario u) throws Exception;
23
24     public Crianca gravaCrianca(Crianca c) throws Exception;
25     public void excluiCrianca(Crianca c) throws Exception;
26     public List<Crianca> listaCriancas() throws Exception;
27     public List<Crianca> listaCriancas(String crianca) throws Exception;
28     public List<Crianca> buscaCriancas() throws Exception;
29
30     public Consulta gravaConsulta(Consulta c) throws Exception;
31     public void excluiConsulta(Consulta c) throws Exception;
32     public List<Consulta> listaConsultaPorCrianca(Crianca c) throws Exception;
33     public List<Consulta> buscaConsultaCriancaGrafico(Crianca c) throws Exception;
34
35     public Usuario login(String login, String senha) throws Exception;
36     public void logout();
37     Usuario getUsuarioDaSessao();
38
39 }
```

Figura 51 - Interface Síncrona

- A interface `RemoteService` é utilizada para identificar quais são os serviços GWT. Com isso o GWT sabe que o código Java criado deve ser traduzido para Javascript;
- A Classe `GreetingServiceImpl` é a implementação da interface `GreetingService` que está situada no cliente. Nela contém a implementação de todos o serviços declarados na interface `GreetingService`. A Figura 52 é o exemplo de uma parte da classe `GreetingServiceImpl` implementada no Servidor.

```
20 @SuppressWarnings("serial")
21 public class GreetingServiceImpl extends RemoteServiceServlet implements
22         GreetingService {
23
24
25     @Override
26     public Usuario gravaUsuario(Usuario u) throws Exception{
27         // TODO Auto-generated method stub
28             UsuarioEntity user = new UsuarioEntity(u);
29              try {
30                  daoUsuario.gravar(user);
31                 return user.toUsuario();
32             } catch (Exception e) {
33                 // TODO Auto-generated catch block
34                 throw new Exception(e.getMessage());
35             }
36     }
```

Figura 52 - Classe GreetingServiceImpl

- A classe `RemoteServiceServlet` da Figura 52 é um *HttpServlet* portanto contém todos os métodos *doGet* e *doPost*, todo serviço implementado em GWT deve ser uma subClasse de `RemoteServiceServlet`;
- Além de criar uma interface síncrona, precisa ser criado uma assíncrona, que serve como um mecanismo de chamada na requisição dos serviços. Para isto basta criar outra interface, que seria uma tradução direta da interface síncrona, os métodos contidos nela devem estar exatamente com o mesmo nome e tipo de retorno, além de adicionado um parâmetro final, que é um `AsynCallback`. Está interface assíncrona serve para que a aplicação não fique "congelada" a espera da reposta quando um serviço for invocado, desta maneira o cliente pode executar outras tarefas na aplicação, a Figura 53 é o exemplo da interface assíncrona da aplicação;

```
13 public interface GreetingServiceAsync {
14
15     public void gravaUsuario(Usuario u, AsyncCallback<Usuario> callback);
16     void listaUsuarios(String usuario, AsyncCallback<List<Usuario>> callback);
17     void excluiUsuario(Usuario u, AsyncCallback<Void> callback);
18     void listaUsuarios(AsyncCallback<List<Usuario>> callback);
19     void buscaUsuariosMedico(AsyncCallback<List<Usuario>> callback);
20     void buscaUsuariosMae(AsyncCallback<List<Usuario>> callback);
21     void gravaCrianca(Crianca c, AsyncCallback<Crianca> callback);
22     void excluiCrianca(Crianca c, AsyncCallback<Void> callback);
23     void listaCriancas(AsyncCallback<List<Crianca>> callback);
24     void buscaCriancas(AsyncCallback<List<Crianca>> callback);
25     void listaCriancas(String crianca, AsyncCallback<List<Crianca>> callback);
26     void gravaConsulta(Consulta c, AsyncCallback<Consulta> callback);
27     void excluiConsulta(Consulta c, AsyncCallback<Void> callback);
28     void listaConsultaPorCrianca(Crianca c,AsyncCallback<List<Consulta>> callback);
29     void buscaConsultaCriancaGrafico(Crianca c,AsyncCallback<List<Consulta>> callback);
30     void login(String login, String senha, AsyncCallback<Usuario> callback);
31     void getUsuarioDaSessao(AsyncCallback<Usuario> callback);
32     void logout(AsyncCallback<Void> callback);
33
34 }
```

Figura 53 - Interface Assíncrona

Como pode ser visto desenvolver com GWT é muito mais trabalhoso. Com Vaadin não é necessário declarar interfaces e implementar classes para a comunicação cliente e servidor, pois sua arquitetura de programação é voltada para o servidor, basta iniciar o desenvolvimento da aplicação no método `init` da classe que estende de `Application`, tornando o desenvolvimento da aplicação muito mais produtivo comparando com GWT.

4.7 LICENÇA DOS FRAMEWORKS

Uma das considerações que são levadas adiante e podem decidir para a escolha de um *framework*, é a questão de seu licenciamento. Além disso, é importante analisar se sua licença gratuita oferece recursos necessários para desenvolver uma aplicação de grande porte.

Tanto Vaadin como GWT são gratuitos e licenciados pela Apache 2.0. No Vaadin existem alguns componentes de terceiros que são gratuitos e podem ser utilizados com o *framework.* O GWT apresenta várias extensões para se utilizar, uma delas é a biblioteca SmartGWT, na qual foi utilizado seus componentes durante o desenvolvimento da aplicação.

A Biblioteca SmartGWT possui licenças pagas, e uma gratuita a *LGPL Edition*, na qual apresenta um conjunto completo de componentes, todas as licenças possuem a mesma quantidade de componentes no lado do cliente, o que muda são os recursos a serem usados no lado do servidor, no qual não houve necessidade de uso.

Como dito acima, os *frameworks* utilizados na aplicação possuem licença gratuita, na qual oferecem recursos necessários para se desenvolver uma aplicação, sem se preocupar em comprar alguma extensão ou mesmo a licença.

4.8 DOCUMENTAÇÃO E ATUALIZAÇÕES

Outro fator levado em extrema consideração para a boa escolha de um *framework* é se ele possui atualizações constantes e uma boa documentação para se trabalhar. Pois se não possui mais investimento para atualizações, a partir do momento que o mercado passar a oferecer soluções modernas e novas técnicas, o *framework* utilizado na aplicação não vai oferecer esses novos recursos. Outra questão é se o *framework* possui uma boa documentação para se trabalhar, pois quando tiver alguma duvida ou problema no decorrer do processo de desenvolvimento é necessário se ter algum material, ou mesmo, uma documentação para apresentar exemplos com recursos no uso do *framework*.

Neste quesito, os *frameworks* Vaadin e GWT apresentam atualizações constantes, e uma ótima documentação, no site do Vaadin é possível baixar o livro com toda sua documentação, e exemplos de como funciona sua arquitetura e seus componentes. A única desvantagem do Vaadin em relação ao GWT é o fato de apresentar pouco material em português. A Biblioteca SmartGWT também possui atualizações constantes, além de apresentar uma comunidade ativa e uma boa documentação.

4.9 COMPONENTES DISPONIVEIS

Um dos fatores importantes para a escolha de um determinado *framework é* se o mesmo possui uma vasta quantidade de componentes para o uso, ou seja, se sua biblioteca consegue resolver todos os problemas encontrados no decorrer do processo de desenvolvimento da aplicação. Outra questão é se esses componentes são customizados de uma maneira produtiva ao desenvolvedor.

O *framework* Vaadin apresenta diversos componentes, a grande vantagem do seu uso é que os componentes são customizados de uma maneira agradável, apresentam uma arquitetura de se trabalhar produtiva.

Com o *framework* GWT, foi utilizado a biblioteca SmartGWT para desenvolver toda a parte gráfica da aplicação, pois utilizar GWT puro não traz vantagem alguma, tornando o

desenvolvimento muito trabalhoso. A Biblioteca SmartGWT também possui uma diversidade de componentes para se trabalhar. A diferença é que seus componentes não são produtivos comparando com os do *framework* Vaadin, e são menos agradáveis para o usuário final comparando com Vaadin.

5 CONSIDERAÇÕES FINAIS

O trabalho desenvolvido foi baseado no estudo de dois *frameworks* para desenvolvimento *Web*. Foram desenvolvidas duas aplicações, na qual apresentam as mesmas funcionalidades e com aparência muito semelhante, os *frameworks* utilizados foram Vaadin e GWT com a biblioteca SmartGWT.

5.1 CONCLUSÃO

Com o desenvolvimento dessas duas aplicações foi possível realizar uma análise comparativa entre os dois *frameworks*. Nesta análise, foram demonstradas as partes mais importantes enfrentadas no decorrer do processo de desenvolvimento, apontando quais as vantagens e desvantagens de cada um.

Através da análise dos fatores, chega-se à conclusão de que o *framework* Vaadin é melhor em vários aspectos comparando com GWT e a extensão SmartGWT. No desenvolvimento de formulários, tabelas e validação de dados, possui uma arquitetura de programação muito mais produtiva, com Vaadin é permitido criar tabelas e formulários através de um Data Model que é ligado com a classe criando todos os campos automaticamente. Com SmartGWT é preciso criar todos os campos tornando o processo de desenvolvimento muito trabalhoso, utilizando muito mais classes e código para criar uma simples formulário ou uma simples tabela.

Outra desvantagem do GWT é o fato de apresentar uma arquitetura robusta, ou seja, se precisar fazer uma pequena tarefa com o *framework*, se torna muito trabalhoso e pouco produtivo comparando com Vaadin. Com GWT é preciso criar classes e interfaces para a comunicação cliente e servidor, já com Vaadin isso não é necessário.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

O presente trabalho demonstrou como é se trabalhar com um *framework* para desenvolvimento *web* que foca apenas na linguagem Java, para trabalhos no futuro abriu as considerações para pesquisar mais *frameworks* que trabalham dessa maneira, outra questão é

que o GWT possui diversas extensões para se utilizar, um ponto importante seria realizar um comparativo entre estas extensões. Outra questão é adicionar mais funcionalidades para aplicação desenvolvida, por exemplo, a mãe da criança ter acesso ao sistema e verificar como foram as consultas de seu filho.

6 REFERÊNCIAS BIBLIOGRÁFICAS

BALZER, C. E.; BATALHA, G.; FERRASA, A. ESTUDO COMPARATIVO ENTRE FRAMEWORKS PARADESENVOLVIMENTO DE RIAs. **ESTUDO COMPARATIVO ENTRE FRAMEWORKS PARADESENVOLVIMENTO DE RIAs**, 2010.

CAELUM. Desenvolvimento Ágil para a. In: CAELUM **Desenvolvimento Ágil para a**., 2011. p. 85-86.

CÉSAR EDENIR BALZER, G. B. A. F. ESTUDO COMPARATIVO ENTRE FRAMEWORKS PARADESENVOLVIMENTO DE RIAs. **ESTUDO COMPARATIVO ENTRE FRAMEWORKS PARADESENVOLVIMENTO DE RIAs**, 2010.

CHU-CARROL, M. C. **Código na Nuvem Programação do Google App Engine**. [S.l.]: Ciência Moderna, 2012.

GONÇALVES, D. B.; VAHL, J. C. J. Web 2.0 – Frameworks de desenvolvimento. **Web 2.0 – Frameworks de desenvolvimento**, 2010.

MAGALHÃES, É. Globalcode. **Globalcode**, 07 jul. 2009. Disponivel em: <http://www.globalcode.com.br/noticias/EntrevistaGWT>. Acesso em: 12 dez. 2012.

MELLO, M. PUCPR. **PUCPR**, 2010. Disponivel em: <http://www.las.pucpr.br/mcfmello/BD/BD-Aula02-MER.pdf>. Acesso em: 18 fev. 2013.

MOTA, L. C. frameworkdemoiselle. **frameworkdemoiselle**, 05 maio 2010. Disponivel em: <http://www.frameworkdemoiselle.gov.br/menu/framework/sobre/>. Acesso em: 12 dez. 2012.

NOGUEIRA, A. Linha de Código. **Linha de Código**, 2011. Disponivel em: <http://www.linhadecodigo.com.br/artigo/987/uml-unified-modeling-language-esteriotipo-include-esteriotipo-extend-esteriotipo-realize.aspx>. Acesso em: 17 fev. 2013.

ORACLE. Oracle. **Oracle**, 2010. Disponivel em: <http://www.oracle.com/technetwork/java/index-jsp-135475.html>. Acesso em: 06 dez. 2012.

PEREIRA, H. C. Revolucao. **Revolucao**, 31 ago. 2005. Disponivel em: <http://revolucao.etc.br/archives/o-que-e-quirks-mode/>. Acesso em: 12 dez. 2012.

PINHEIRO, G. UOL. **UOL**, 18 mar. 2011. Disponivel em: <http://webinsider.uol.com.br/2011/03/18/cinco-ferramentas-para-a-producao-de-wireframes/>. Acesso em: 19 fev. 2013.

PRADA, R. tecmundo. **tecmundo**, 2008. Disponivel em: <http://www.tecmundo.com.br/hardware/210-o-que-e-plugin-.htm>. Acesso em: 01 mar. 2013.

SMARTCLIENT. SmartClient. **SmartClient**, 2008. Disponivel em: <http://www.smartclient.com/technology/>. Acesso em: 20 fev. 2013.

SMEETS, B.; BONESS, U.; BANKRAS, R. **Programando Google Web Toolkit Do Iniciante ao Profissional**. [S.l.]: ALTA BOOKS, 2009.

SOUZA, M. V. B. D. Estudo Comparativo entre Frameworks Java. In: SOUZA, M. V. B. D. **Estudo Comparativo entre Frameworks Java**. [S.l.]: [s.n.], 2004. p. 6-11.

TERRA. Terra. **Tera**, 2007. Disponivel em: <http://www.terra.com.br/informatica/ajuda/seguranca/tipos_script.htm>. Acesso em: 01 fev. 2013.

UFCG. http://www.dsc.ufcg.edu.br. **http:** //www.dsc.ufcg.edu.br, 2010. Disponivel em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/diagramas/interacao/sequencia.htm>. Acesso em: 18 fev. 2013.

ULLMAN, C. **Site da wrox**. Disponivel em: <http://www.wrox.com/WileyCDA/Section/What-is-Ajax-.id-303217.html>. Acesso em: 01 dez. 2012.

VAADIN. **Book of Vaadin:** 4th Edition. [S.l.]: [s.n.], 2012.

APÊNDICES

APÊNDICE A

Para instalar o *framework* Vaadin no eclipse, acesse o menu *Help/Install new software* coloque a URL http://vaadin.com/eclipse, conforme Figura 54.

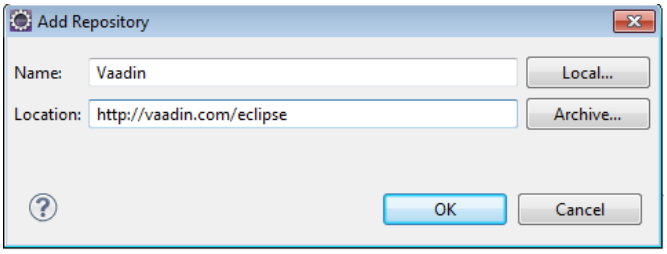


Figura 54 - Instalando Vaadin

Após isso selecione o *pluguin* do Vaadin e finalize a instalação, conforme Figura 55.

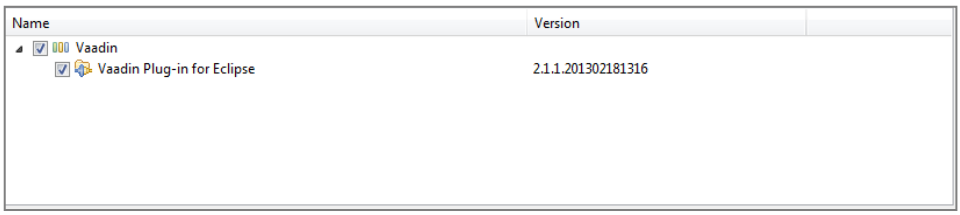


Figura 55 - Selecionando o *pluguin*

Para criar um projeto no Vaadin é preciso:

- Primeiramente, selecione o menu File, new Project.
- Após ir a new Project, pesquise a pasta Vaadin e selecione Vaadin Project.

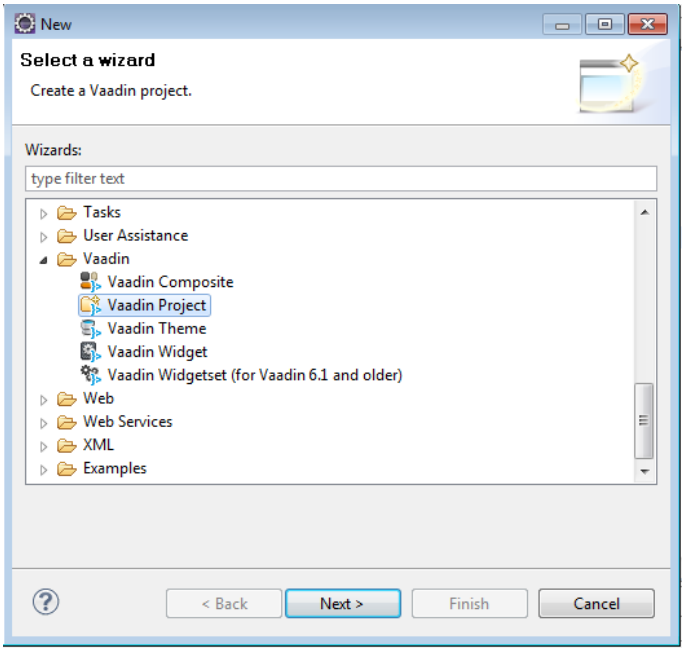


Figura 56 - Criando um Projeto Vaadin

- Na parte de criar o projeto Vaadin, é preciso definir as configurações básicas do projeto *Web*. No Quadro 6, explicação do que serve cada campo da Figura 57.

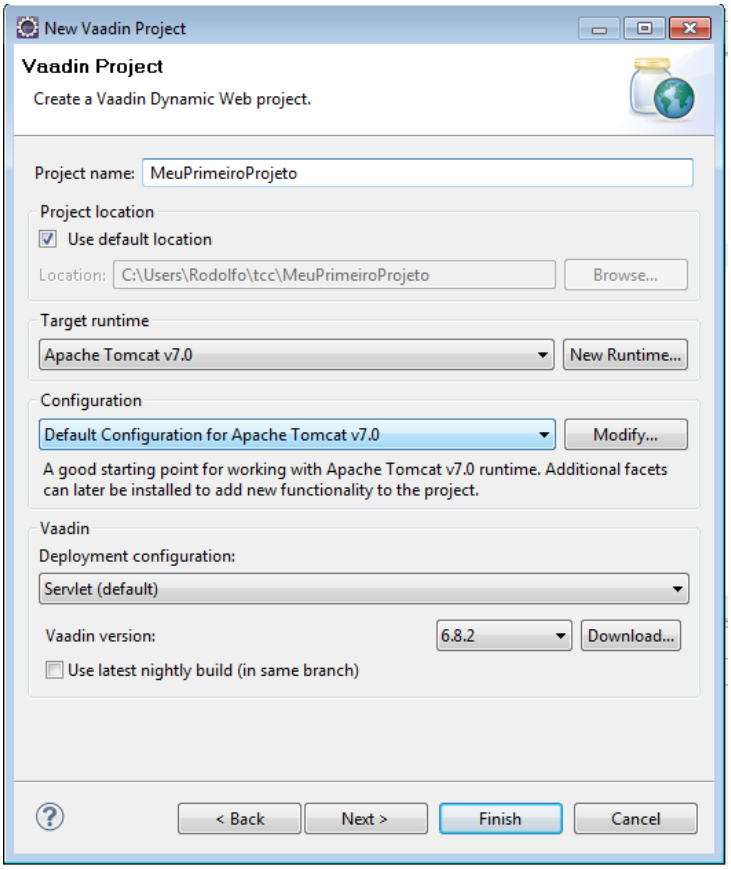


Figura 57 - Configuração Projeto

Valor	Descrição
Project Name	Informa o nome desejado para o Projeto
Use Default	Define o diretório padrão para o Projeto, normalmente se deixa com o valor pré-estabelecido.
Target runTime	Define qual servidor de aplicação será utilizado no projeto.
Configuration	Define a configuração a utilizar, normalmente deve-se usar a configuração padrão do servidor de aplicação.
Deployment Configuration	Define o ambiente no qual o aplicativo será implantado
Vaadin Version	Selecione a versão do vaadin a utilizar, pode ser realizado o *download* da ultima versão

Quadro 6 - Descrição dos Valores na Criação de um Projeto Vaadin

- A próxima etapa é configurar o *Web* Module, que é as configurações relacionada à estrutura do projeto *Web*. Normalmente, deve-se aceitar as configurações padrões estabelecidas, no Quadro 7 uma breve descrição de cada valor da Figura 58.

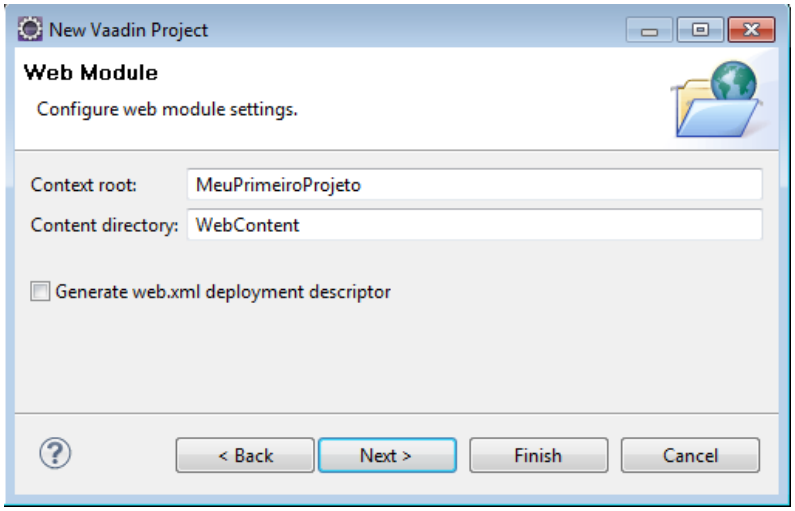


Figura 58 - *Web* Module

Valor	Descrição
Context Root	Identifica qual URL a aplicação vai acessar quando executada, por padrão o nome da URL é o mesmo que o nome do projeto.
Content Diretory	Diretório que contém todo o conteúdo a ser incluído no *servlet* e servido pelo servidor *Web*
Generate *Web*.xml deployment descriptor	Informa se deseja que seja criado o arquivo *Web*.xml, descritor necessário para executar o *servlet*

Quadro 7 - Descrição dos Valores Web Module

- A ultima etapa são as configurações específicas da aplicação, essas configurações podem ser mudadas no decorrer do desenvolvimento do projeto, exceto a configuração de *portlet*. O Quadro 8 descreve os valores da Figura 59.

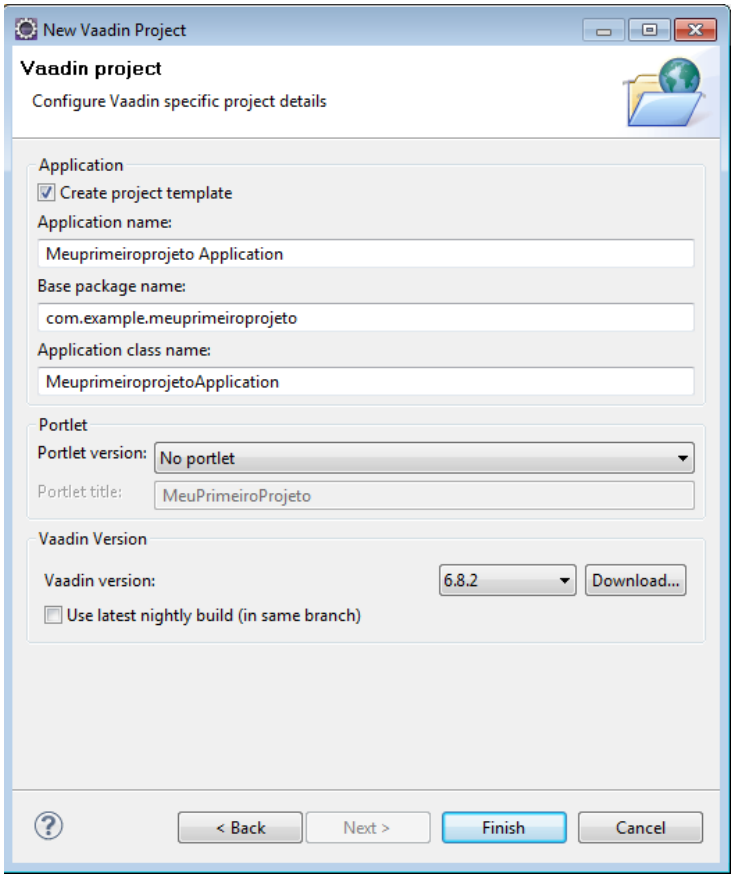


Figura 59 - Configuração Específica

Valor	Descrição
Create Project Template	Define se quer ou não que crie a aplicação com um exemplo.
Application Name	Nome para ser exibido no navegador quando a aplicação for executada, por padrão é selecionado o mesmo nome do projeto.
Base Package name	O nome do pacote em que a classe de exemplo será criada.
Application class name	Nome da classe de exemplo a ser criada
Portlet	Quando essa opção for selecionada, o assistente criará os arquivos necessários para executar o aplicativo em um Portlet.

Quadro 8 - Configuração Específica

Quando terminado todas essas configurações, clique em finish e será criada toda a estrutura do projeto, e também um pequeno exemplo de uma aplicação.

APÊNDICE B

Para instalar o *framework* GWT no eclipse, verifique qual a versão do eclipse está sendo utilizado, no caso desse exemplo é a *IDE Eclipse SDK* na versão *Helios Release 3.* Acesse o menu *Help/Install new software* coloque a URL http://dl.google.com/eclipse/plugin/3.6, conforme Figura 60.

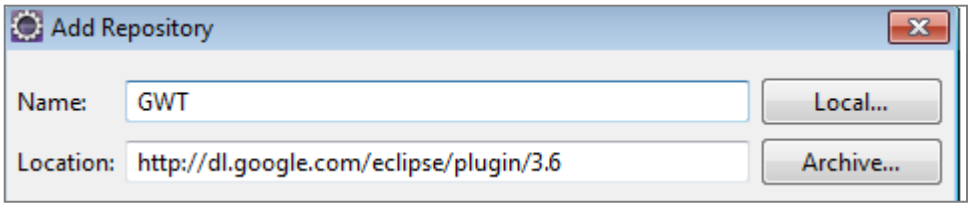


Figura 60 - Instalando GWT

Após isso selecione o *pluguin* do GWT e finalize a instalação, conforme Figura 61.

Figura 61 - Selecionando pluguin GWT

Para criar um projeto com GWT realize os seguintes passos:

- Para criar um novo projeto, primeiramente, acesse o menu File >> new;
- Pesquise a pasta Google, e clique em *Web* Application Project.

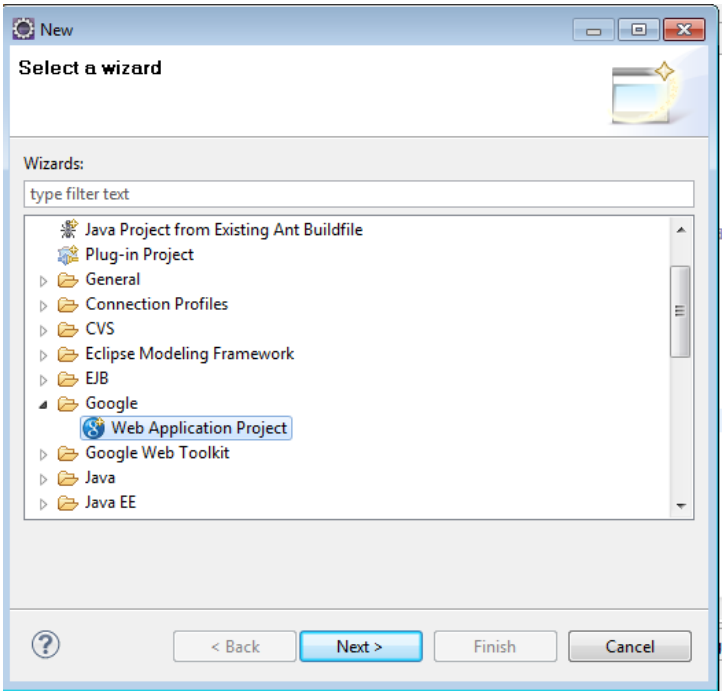


Figura 62 - Criando um Projeto GWT

- A próxima etapa é a configuração da aplicação, a Figura 63 é o exemplo da criação do projeto, no Quadro 9 descreve os valores da Figura 63;

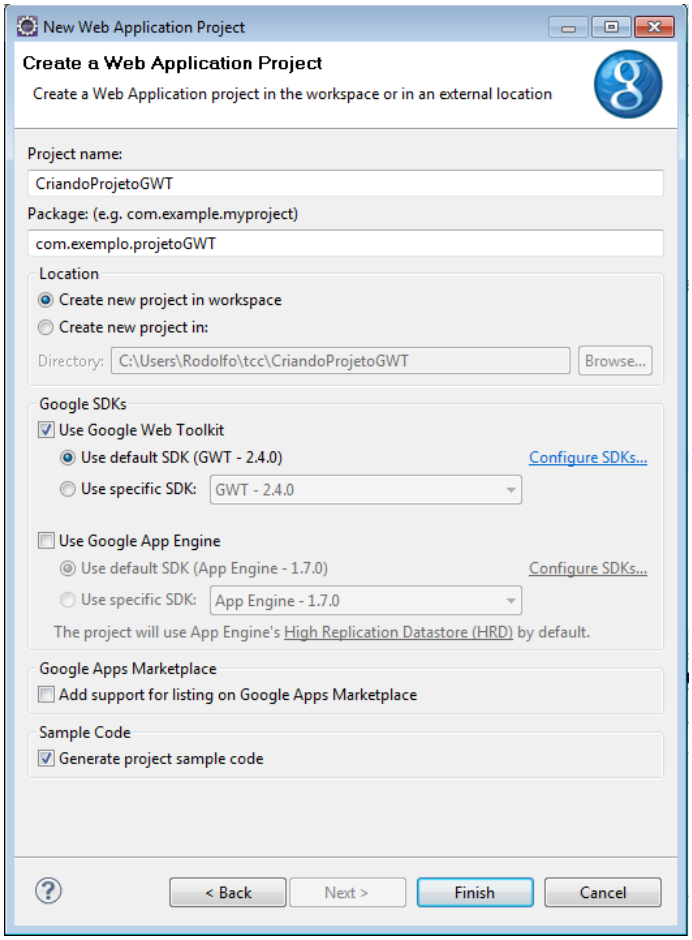


Figura 63 - Configura a Aplicação GWT

Valor	Descrição
Project Name	Nome do Projeto
Package	Pacote onde estará a classe raiz da aplicação
Location	Localização do Projeto
Google SDKs	Define qual GWT será utilizado e qual *engine* deverá ser usado
Sample Code	Será criado uma aplicação de exemplo

Quadro 9 - Configurando a aplicação GWT

- Após isso, clique em finish e será criada uma aplicação de exemplo, para executa-la basta clicar com o botão direito ao lado do projeto e acessar *run* >> *Web application*, a Figura 64 é o exemplo da aplicação.

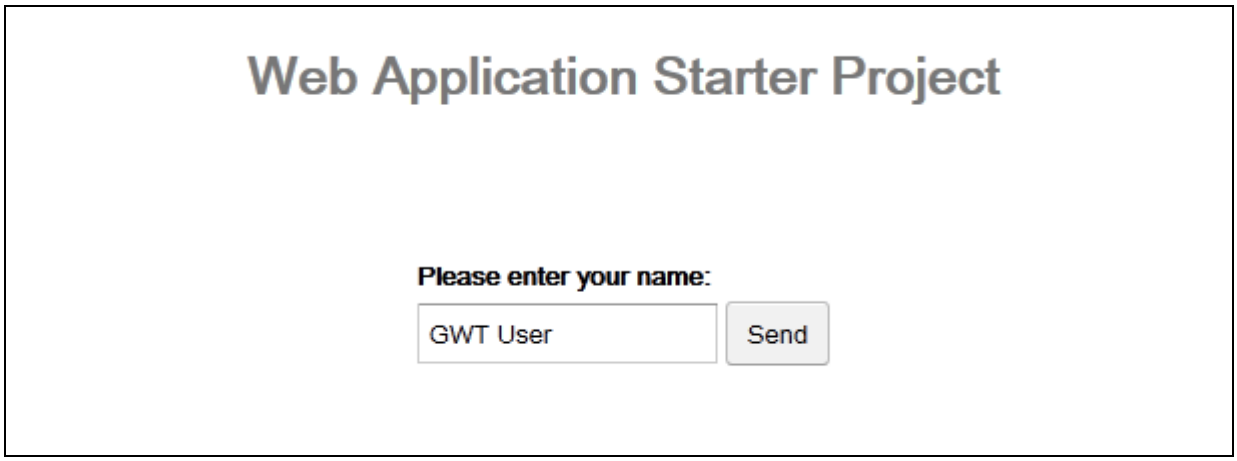


Figura 64 - Aplicação de Exemplo