

Documento de arquitetura - Chamada Parlamentar 2

Desenho de Software

1-RAFAEL FAZZOLINO - 11/0136942
2-THIAGO KAIRALA - 12/0042916
3-EDUARDO BRASIL MARTINS - 11/0115104
4-THABATA GRANJA - 09/0139658

1-fazzolino29@gmail.com
2-thiagor@gmail.com
3-brasil.eduardo1@gmail.com
4-thabatahelen@gmail.com

Brasília, DF - 2015

Histórico de Alterações

Sigla	Significado
V	Versão
MF	Número de arquivos modificados.
AL	Número de linhas adicionadas.
DL	Número de linhas deletadas.

V	Autor	Data	Mensagem do Commit	MF	AL	DL
0	Thiago kairala	2015-03-22	Preparando documento	44	3339	0
1	Thiago kairala	2015-03-22	consertando capa do documento	11	151	136
2	Thiago kairala	2015-03-22	Corrigindo nome dos integrantes	1	6	7
3	Thiago kairala	2015-03-23	Finalizado representação da arquitetura	9	109	88
4	Eduardo	2015-03-22	Alterando o nome do projeto no documento	1	1	1
5	Thiago kairala	2015-03-23	Finalizado metas e restrições de arquitetura	7	20	15
6	Eduardo	2015-03-22	Alterando o nome do projeto no documento	5	11	10
7	Thiago kairala	2015-03-23	Finalizada visao de casos de uso	5	74	19
8	Thiago kairala	2015-03-23	Alterado esquema de arquitetura	14	3	254
9	Thiago kairala	2015-03-24	Finalizada visao de processos	19	339	80
10	Thiago kairala	2015-03-24	finalizado visao de implementação	12	37	68
11	Thiago kairala	2015-03-24	Finalizado parte de qualidade e assim o documento	5	8	6
12	Thiago kairala	2015-03-23	Finalizada visao de casos de uso	4	65	0
13	Rafael Fazzolino	2015-03-30	Problemas das matriculas e e-mails corrigidos	1	8	7
14	Rafael Fazzolino	2015-04-01	Corrigindo erros e retirando conflitos	8	32	102
15	Thabata Helen	2015-04-04	Alterando diagrama de arquitetura	7	22	81

Sumário

1	Introdução	1
1 .1	Finalidade	1
1 .2	Escopo	1
2	Representação da Arquitetura	1
3	Metas e Restrições de Arquitetura	2
4	Visão de Casos de Uso	2
4 .1	Realizações de Casos de Uso	2
5	Visão Lógica	2
5 .1	<i>view</i>	2
5 .2	<i>controller</i>	2
5 .3	<i>model</i>	2
5 .4	<i>dataParser</i>	3
5 .5	<i>dao</i>	3
5 .6	<i>webServiceConnector</i>	3
6	Visão de Processos	3
7	Visão de Dados	3
8	Tamanho e Desempenho	3
9	Qualidade	3

1 Introdução

1.1 Finalidade

Este documento apresenta uma visão geral abrangente da arquitetura do sistema e utiliza uma série de visões arquiteturais diferentes para ilustrar os diversos aspectos do sistema. Sua intenção é capturar e transmitir as decisões significativas do ponto de vista da arquitetura que foram tomadas e criar subsídio para o entendimento dessas decisões.

1.2 Escopo

O sistema "Chamada Parlamentar 2" compreende a consulta de pontualidade e assiduidade dos parlamentares nas sessões do parlamento. Para o contexto do projeto foi fundamental investir um tempo considerável no desenvolvimento da arquitetura, pois a mesma depende de um serviço externo, como o *web service* da camara dos deputados.

Outro detalhe extremamente importante é a modularidade do sistema, para permitir que com obtenção de novos dados, seja possível adicionar novas funcionalidades no sistema. Facilitando a manutenibilidade do mesmo.

2 Representação da Arquitetura

A representação da arquitetura para este projeto pode ser visualizada por meio de um diagrama apresentado na Figura 1.

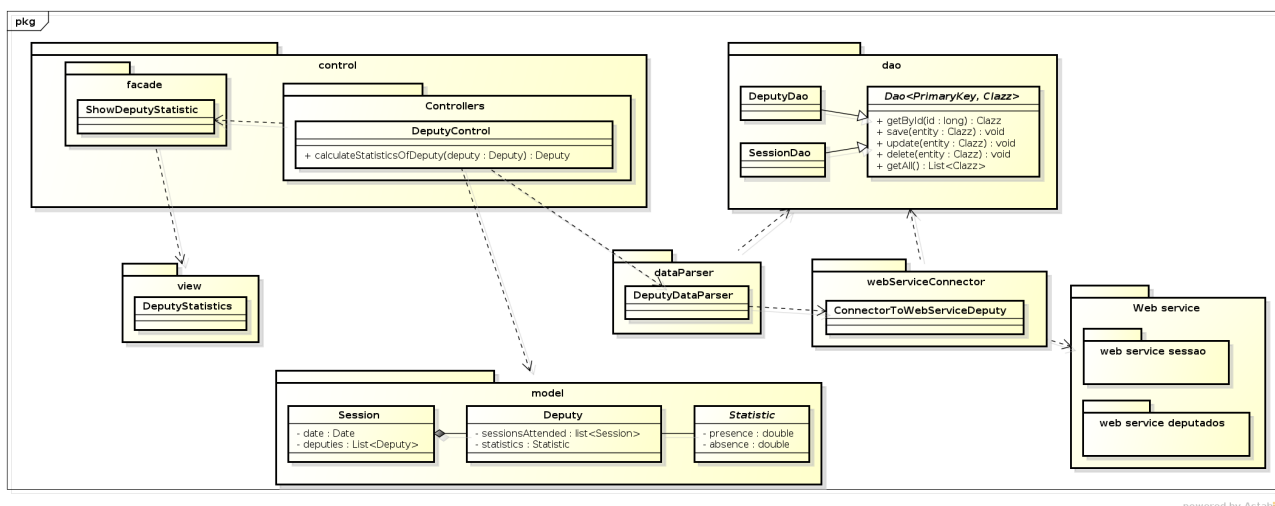


Figura 1. Diagrama da arquitetura do sistema

A arquitetura do *Chamada Parlamentar* foi projetada com o objetivo de tornar o *software* leve e eficiente. A confiança nos dados apresentados pelo sistema se dá pela utilização de duas formas de obtenção dos dados em momento de execução. A primeira forma, e mais atualizada, ou seja, mais confiável, é a disponibilização dos dados via *WebService*.

Caso haja algum problema na conexão com o *WebService*, o sistema fará a obtenção dos dados via Banco de Dados, onde estão armazenados os dados do *WebService* do dia anterior. O sistema fará a atualização do banco de dados todos os dias às 5 horas da manhã.

O pacote *WebServiceConnector* será utilizado para realizar a conexão com o *WebService* e o *dataParser* irá tratar os dados da forma apropriada para armazenamento dos mesmos no banco ou disponibilização dos mesmos, ou seja, estes dois pacotes sempre irão trabalhar lado a lado.

O *Dao* será o pacote que fará a conexão e todas as manipulações necessárias no Banco de Dados. Tanto para edição de dados quanto para obtenção dos mesmos no Banco de Dados.

Os pacotes utilizados para a manipulação dos dados em nível de Base de Dados são o grande diferencial desta arquitetura. O restante é o que forma o conhecido *MVC*, a *modelo*, a *view* e a *controler*.

3 Metas e Restrições de Arquitetura

Durante o desenvolvimento da arquitetura foram levados em conta diversos aspectos como por exemplo a segurança do sistema, a possibilidade de diversas verificações para garantir sucesso e evitar o famoso "*garbage in, garbage out*", a possibilidade de o sistema vir a crescer e por último possibilitar pessoas externas a se identificarem com o sistema e contribuir com novas funcionalidades.

A maior preocupação existente na arquitetura projetada foi a formatação dos dados vindos do web service, pois existe a possibilidade de haver a mudança e não é de controle da equipe de desenvolvimento, por isso fazer um sistema modularizado para facilitar a troca da entrada de dados caso seja necessário.

4 Visão de Casos de Uso

A parte de maior risco arquitetural é a parte de acesso ao *web service* então esta sessão descreverá como será a interação da camada controller com este módulo do sistema

4.1 Realizações de Casos de Uso

Pesquisar um parlamentar

1. Ao receber a requisição a camada da *view* irá pedir a *controller* para que a mesma faça as contas da estatística;
2. A *controller* irá pedir os dados para a camada de *dataParser* para que consiga os dados e entregue-os em um formato esperado;
3. A classe parser irá tentar buscar os dados do *web service* pelas classes presentes no pacote *webServiceConnector*, caso funcione irá retornar os dados para a controller no formato em que a mesma já espera;
4. Se por algum motivo a classe parser tiver alguma dificuldade em recuperar estas informações, será feita então uma requisição a camada *dao* pelas informações no banco de dados;
5. Tendo os dados em mão o sistema irá calcular as estatísticas e finalmente enviá-los para a *view* para que possam ser mostrados ao usuário.

5 Visão Lógica

O sistema Chamada Parlamentar 2 será executado via web em um browser, que por sua vez fará requisições ao sistema no servidor. Nesta sessão será explicada cada uma das camadas que estão sendo executadas no servidor de cima para baixo.

5.1 *view*

Esta é a primeira das camadas encontradas após ser feita uma requisição para o sistema. Consiste apenas de uma conexão simples entre as páginas do *browser* e as classes java por trás do sistema.

Geralmente estas classes serão simples e extendidas da classe *HTTPServlet* do java que é o framework utilizado durante este projeto. Estas classes serão responsáveis por chamar as *controllers* para que estas realizem as ações.

5.2 *controller*

As classes do pacote *controller* são responsáveis por toda a lógica do sistema. Serão sempre instanciadas por uma *view*, e apenas poderão acessar as classes no pacote *dataParser* para ter uma interface simples caso seja necessário alterar o método de entrada.

5.3 *model*

A classes presentes no pacote *model* são as classes mais simples do sistema porem de grande importância já que elas são responsáveis por ter as entidades representadas no sistema.

Estas classes poderão ser instanciadas pelas classes nos pacotes *view*, *controller* e *dataParser*.

Por serem as classes com maior conexão com o resto do sistema, as *models* são um ponto de risco para a arquitetura por isso será gasto um tempo considerável desenvolvendo cada um dos diagramas de cada uma delas para garantir que não será necessário alterar muita coisa caso haja uma mudança.

5.4 *dataParser*

As classes existentes no pacote *dataParser* são responsáveis por prover às *controllers* as informações em um formato esperado. Estas informações podem ser encontradas ou no banco de dados ou no *web service* da camara dos deputados estando cada um destes em um formato diferente.

Sempre que for necessária uma informação será primeiro requisitado ao *web service*, apenas se não for possível a utilização do mesmo será usado o banco e dados.

Os dados do banco de dados são acessados pelas classes existentes no pacote *dao*, enquanto os dados vindos do *web service* são trazidos pelas classes existentes no pacote *webServiceConnector*.

5.5 *dao*

As classes implementadas na camada *dao* são responsáveis pelo *C.R.U.D.* de todas as informações no banco de dados, garantindo assim que se o banco de dados sofrer alguma alteração não será necessário mexer no código de nenhuma outra classe, apenas no pacote *dao*.

5.6 *webServiceConnector*

Este pacote contém as classes que irão fazer a conexão com o *web service* para buscar as informações solicitadas pelas classes do pacote *dataParser*.

Este é o pacote de maior risco arquitetural do projeto pois o *web service* não depende da equipe e caso haja uma alteração será necessária uma mudança drástica neste pacote e por isso foi isolado totalmente do resto do sistema.

6 Visão de Processos

Para deixar o processo de cada um dos pacotes o mais independente possível sempre que houver a necessidade de se utilizar dados de uma classe em outra estes dados serão passados por parametros de métodos, evitando assim acoplamento desnecessário entre as classes.

Para as classes do pacote *webServiceConnector* haverá uma diferença pois as informações oriundas da Camara dos deputados é recebida em *xml*. A classe deverá passar as informações para objetos java e então retornar os objetos prontos para serem utilizados pelas *controllers*.

7 Visão de Dados

ADICIONAR O MER AQUI E EXPLICÁ-lo

8 Tamanho e Desempenho

Para o sistema funcionar perfeitamente é necessário que as consultas no *web service* sejam rápidas, assim é necessário que as informações trazidas sejam limitadas ao período da eleição do deputado até o dia atual, totalizando assim no máximo 4 anos de dados de sessões.

9 Qualidade

A arquitetura proposta neste documento impacta diretamente em diversos quesitos no funcionamento do software, como por exemplo extensibilidade, confiabilidade, e a capacidade de ser alterado um módulo sem comprometer outros pacotes, o que permite até se for o caso a troca no meio em que o *web service* é consumido em algum ponto futuro.

Referências Bibliográficas