Universidade do Minho

Mestrado em Engenharia Informática

Arquiteturas Aplicacionais

Ano lectivo 2014/2015

Trabalho Prático

Comparação de Frameworks

Autores:
José Morgado (pg27759)
Luís Miguel Pinto (pg27756)
Pedro Carneiro (pg25324)

Braga, 3 de Março de 2015

Resumo

Neste trabalho serão identificadas e analisadas *frameworks* respeitantes a cada uma das três camadas (dados, lógica e apresentação) do modelo de três camadas.

A intenção passa por escolher, para cada uma das camadas referidas anteriormente, a que mais se adequa aos interesses de um programador que pretenda desenvolver uma aplicação.

Conteúdo

1 Introdução

O desenvolvimento de aplicações e as metodologias aplicadas a este processo sempre foram alvo de estudo cuidado e intensivo por parte da comunidade de engenheiros de *software*.

Um dos modelos deste paradigma que mais aceitação reuniu na comunidade foi o modelo de três camadas (*Three Tier Architecture*). Este defende a separação clara no desenvolvimento da aplicação de conceitos distintos: a **interface com o utilizador**, a **lógica de negócio** e os **dados**. Assim, neste modelo, cada um destes conceitos terá mapeamento direto na sua camada respetiva:

Camada de Apresentação É a camada responsável pelo desenvolvimento da interação entre o sistema e o utilizador. É nesta camada que os pedidos dos clientes serão inicialmente processados e, também, será nesta onde serão exibidos os resultados a estes dos seus pedidos.

Camada de Negócio É nesta camada que se encontra a solução desenhada para o problema. Esta apresenta propriedades de dados voláteis. Assim, por forma a persisti-los, terá de comunicar com a camada de dados.

Camada de Dados Esta camada é responsável por armazenar os dados do problema. Não pode, por isso, ser desassociada a uma base de dados.

O objetivo deste trabalho passa por identificar e analisar, para cada uma destas camadas, primeiro individual e depois globalmente, *frameworks* no âmbito da linguagem de programação *Java* que possam facilitar o processo de desenvolvimento de *software*.
Assim, para cada camada, e mediante a opinião do grupo, será indicada a *framework* que melhor serve os interesses do programador.

2 Camada de Dados

A camada de dados funciona como uma interface para persistir os dados/objetos manipulados pela camada de negócio. Esta interação entre as camadas pode ser baseada numa abordagem centrada em qualquer uma delas. Temos por isso de um lado as frameworks ORM, como o Hibernate, e por outro lado as frameworks que colocam a base de dados em primeiro lugar, como o jOOQ. Apresentaremos ainda a biblioteca DbUtils, uma alternativa que simplifica a utilização do JDBC.

2.1 Hibernate

O Hibernate é a framework ORM mais popular. O mapeamento entre objetos e os dados existentes nas tabelas da base de dados pode ser configurado através de um ficheiro XML ou Java Annotations.

Apresenta as seguintes vantagens:

- **Produtividade** Permite persistir o estado dos objetos de forma muito simples, não sendo sequer necessário conhecimentos de SQL.
- **Manutenibilidade** Permite reduzir o número de linhas de código, pelo que é mais fácil perceber o sistema e efetuar refactoring.
- **Portabilidade** Facilmente se muda de base de dados, bastando proceder a algumas mudanças no ficheiro de configuração.

A principal desvantagem tem que ver com o impacto em termos desempenho.

2.2 jOOQ

Esta ferramenta permite gerar código Java a partir da base de dados e escrever código Java com uma sintaxe mais próxima do SQL. Por exemplo:

- **SQL**

```
SELECT * FROM BOOK
WHERE BOOK.PUBLISHED_IN = 2011
ORDER BY BOOK.TITLE
```

- **Java**

```
create.selectFrom(BOOK)
      .where(BOOK.PUBLISHED_IN.eq(2011))
      .orderBy(BOOK.TITLE)
```

Surgiu como resposta para algumas das desvantagens inerentes ao uso de frameworks ORM. Alguns dos seus pontos fortes são:

- **Base de Dados em Primeiro Lugar** Permite utilizar todas as capacidades dos RDBMs e do SQL.

- **Typesafe SQL** A linguagem SQL é type safe, e essa característica é mantida no código Java.
- **Active Records** Usando este padrão, não é necessário escrever comandos SQL para lidar com as operações do tipo CRUD.
- **Standadização** Permite utilizar qualquer dialeto do SQL, pois as expressões são adaptadas automaticamente à base de dados utilizada. Assim, o mesmo código funcionará em qualquer base de dados.
- **Procedures** O gerador de código também gera métodos para cada procedimento.

2.3 DbUtils

Esta biblioteca foi concebida com o intuito de facilitar a utilização do JDBC, reduzindo o código necessário para efetuar queries e updates. Apresenta as seguintes vantagens:

- Redução da probabilidade de ocorrência de erros inerentes à gestão de recursos/ligações.
- Código mais limpo e legível.
- Geração automática de JavaBeans a partir dos ResultSets.

2.4 Selecção de uma alternativa

Todas as alternativas têm as suas vantagens e desvantagens, pelo que a escolha deve ser ponderada consoante o caso:

- **Hibernate** Se quisermos maior produtividade e abstrair ao máximo a camada de dados.
- **jOOQ** Se pretendermos tirar partido das vantagens do SQL, esta alternativa possibilita fazê-lo diretamente em Java de forma rápida e segura.
- **DbUtils** Se o projeto em causa for de dimensão reduzida e se apenas se pretender uma alternativa ao JDBC.

3 Camada Lógica

A camada lógica, também designada por camada de negócios, situa-se na posição intermédia no modelo de três camadas abordado e estudado neste relatório. Analisando este modelo sobre uma perspetiva *bottom-up*, a camada de negócios (*layer II*) surge depois da camada de dados (*layer I*)e é sucedida pela camada apresentacional (*layer III*).

O *software* desenvolvido para esta camada é executado, apenas e só, no servidor da aplicação. Num modelo de desenvolvimento de uma aplicação seguindo o modelo de três camadas, é esperado que a camada lógica assegure os seguinte objetivos:

- Definição das Classes do Problema
- Definição da Lógica Aplicacional
- Validação das Regras de Negócio

Como referido no primeiro ponto, o desenho da solução do problema ocorre, quase por exclusivo, nesta camada. Assim, e uma vez que estamos num ambiente de programação orientada aos objetos, e, em particular, recorrendo à linguagem *Java*, é nesta camada que serão desenvolvidas os *packages* e as classes que representam as entidades do domínio do problema. É também nestas classes que estará explícita as regras de negócio constituintes da aplicação e onde também será feita as validações ao modelo adotado.

O facto desta camada ser a camada intermédia faz com que desempenhe também um papel fundamental na comunicação entre as outras duas camadas. Assim, a camada de negócios é responsável por receber e processar os pedidos que a camada de interface com o utilizador lhe faz chegar e, por sua vez, comunicar com a camada de dados caso seja necessária atualizar o estado da aplicação.

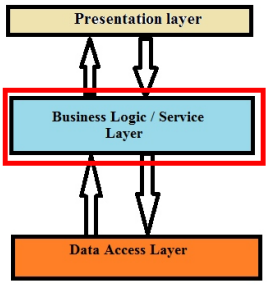

Figura 1: Modelo das três camadas com enfâse na camada lógica

Devido a esta interoperabilidade permanente, facilmente se verifica que esta camada arcará com a maior parte do processamento da aplicação, pelo que otimizações ao código desenvolvido para resultam, em grande parte das vezes, num ganho significativo a nível de *performance* da aplicação.

Pode-se então afirmar, com certeza, que é nesta *Business Layer* que se encontra o *core* da aplicação.

Importa agora enquadrar as *frameworks* no contexto da camada que está a ser analisada. As *frameworks* surgem com o objetivo de, essencialmente, facilitar o processo de desenvolvimento de *software* e, assim, tornar mais eficiente e produtivo a produção da aplicação. Estas permitem ao programador trabalhar a um nível mais alto de abstração, automatizando, e por forma a fornecer um exemplo, o processo de *queries* à base de dados, que, como se sabe, é uma tarefa algo repetitiva.

Para a análise comparativa destas ferramentas para a camada de negócio foram selecionadas duas *frameworks*: ***Enterprise JavaBeans*** e ***Spring***. O estudo de comparação assentará em critérios como *(i)* o tempo necessário para dominar a ferramenta, *(ii)* a portabilidade, *(iii)* a integração de testes unitários e *(iv)* a documentação disponível fornecida tanto por fonte oficial como pela comunidade.

3.1 *Enterprise JavaBeans Technology*

A *Enterprise JavaBeans*, também conhecida como *EJB*, é uma *framework* responsável pela encapsulamento da componente, por parte do servidor, da lógica de negócio de uma dada aplicação. A especificação original foi inicialmente desenvolvida pela *IBM*, sendo adotada, mais tarde, pela *Sun Microsystems*. Desde então, esta especificação tem sido mantida e melhorada pelo formalismo *Java Community Process*.

O objetivo desta tecnologia é colocar ao dispôr do programador uma forma standardizada de desenvolver a lógica de *back-end* de uma aplicação *web*. Este desenvolvimento é feito num nível mais alto de abstração, uma vez que é a própria *framework* que se encarrega das tarefas como a persistência da informação na base de dados, a garantia de integridade das transações e da segurança, permitindo que a equipa de desenlvovimento se foque na arquiterua desenvolvida para dar resposta ao problema e na lógica de negócio associada.

No que se refere aos critérios de comparação, e começando no tempo que será necessário dispender para dominar a ferramenta, consta-se que pelo facto de fazer parte do standards de *Java* torna o processo adpatativo bem mais curto. Qualquer programador de *Java* se sentirá familiar com a forma como a *API* foi organizada e desenvolvida.

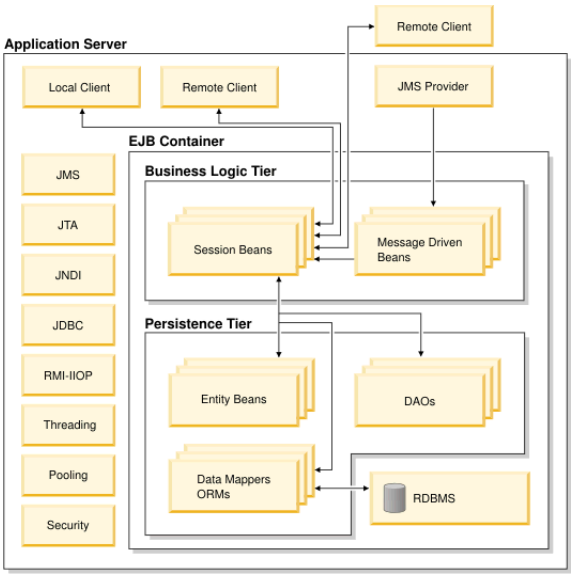

Figura 2: Arquitetura do *EJB*

Relativamente à portabilidade desta *framework*, e por se tratar de *software* produzido para a *Java Virtual Machine*, esta ferramenta deverá ser, à partida, independente da plataforma, significando, portanto, que poderá ser executada em qualquer conjunto de *Hardware+Software* onde a máquina virtual do *Java* seja suportada, o que, atualmente, acontece com a grande parte dos computadores e dispositivos móveis.

Já quanto à realização de testes unitários, esta é possível e é feita através do módulo de *JUnitEE*.

Por último, importa fazer um comentário acerca da documentação existente para esta *framework*. Ora, como é do conhecimento geral, um dos pontes fortes do *Java* é a sua detalhada documentação e, sendo este um standard que é parte constituinte da linguagem, a documentação existente é bastante satisfatória. A comunidade também desempenha um papel importante na hora de adoção a uma *framework* e a desta, incorporada na de Java, também não desilude. Desta forma, não deverá ser difícil encontrar apoio *online* para questões relacionadas com esta tecnologia.

3.2 *Spring Framework*

A *framework Spring* é outra solução a ter em conta no âmbito de desenvolvimento de *software* para a camada de negócios. Esta tem como objetivo lidar da infraestrutura e do servidor aplicacional onde a aplicação é executada, fazendo com que a equipa de desenvolvimento se possa focar primariamente no desenho da arquitetura que visa dar resposta ao problema.

O *Spring* apresenta-se como um framework perfeitamente modular. É composta, assim, por várias componentes e dá total liberdade de escolha ao programador para integrar os módulos que são necessários para a especificidade do problema. A *big picture* da arquitetura do Spring, com todos os seus módulos, é apresentada na figura seguinte.

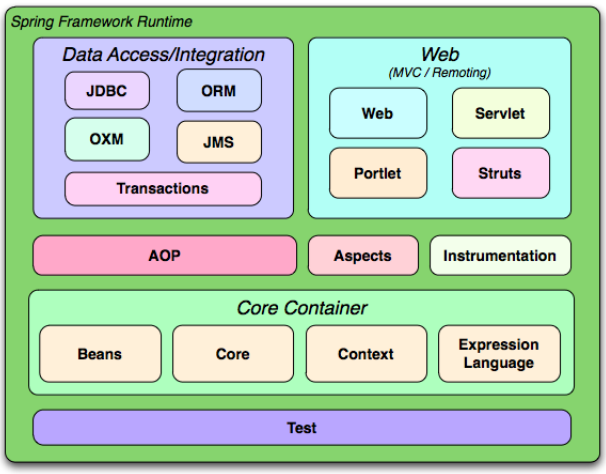

Figura 3: Arquitetura da *Spring Framework*

Das componentes exibidas na figura anterior, apenas são relevantes para este caso de estudo as pertencentes ao módulo *Core Container*.

Efetuando uma pequena análise à facilidade ou não de aprendizagem desta tecnologia, verifica-se que, apesar do vasto leque de recursos existentes, ainda é necessário um tempo de adaptação razoável para lidar com esta *framework*.

A portabilidade não aparenta ser um problema, já que esta corre em cima da máquina virtual do *Java*.

Esta *framework* revela preocupação e incentiva à realização e utilização de testes unitários às aplicações desenvolvidas com esta. Os testes podem também ser do tipo de integração, onde são verificadas e validadas as informações da cache e as meta-anotações presentes nas classes desenvolvidas para a solução do problema.

A documentação da *framework* está bem organizada e são cobertos todos os tópicos. Ao nível da comunidade, o fórum oficial desta ferramenta será a melhor forma para obter respostas a eventuais dúvidas que possam aparecer.

3.3 Comparação Final

As duas *frameworks* apresentadas revelam-se como soluções sólidas, robustas e escaláveis para o desenvolvimento da camada de negócios aplicacional.

No entanto, e havendo a necessidade de se efetuar uma escolha, a opção recai no *EJB*, dado que, por um lado, é um standard do Java e, por outro, apresenta uma comunidade bem maior do que a existente em Spring. Na portabilidade, o *EJB* também se destaca pelas garantias mais seguras de portabilidade comparativamente ao *Spring*.

4 Camada de Apresentação

A camada de apresentação é o espaço onde ocorrem interações entre o utilizador e o serviço disponibilizado. É responsável por receber e apresentar dados ao utilizador. Tem ainda como função traduzir e enviar informação para processamento na camada lógica.

Neste sentido, foram surgindo ao longo dos anos várias *frameworks* com o objetivo principal de acelarar e otimizar o desenvolvimento destes procedimentos, permitindo assim produzir uma interface de utilizador mais eficiente e agradável *(user friendly)* de maneira a produzir o resultado pretendido.

Assim ao longo desta secção iremos abordar e comparar algumas das *frameworks* mais utilizadas, tentando perceber no final qual delas se destaca mais entre elas. Para nos ajudar nesta análise vamos utilizar os seguintes recursos na comparação das *frameworks*:

- Prototipagem rápida da aplicação
- Complexidade
- Facilidade de utilização
- Documentação e Comunidade
- Ecossistema
- Escalabilidade
- Manutenção e Atualizações
- User Experience

4.1 Spring MVC

Dos vários artigos e publicações que fomos consultando conseguímos perceber que esta *framework* é das mais utilizadas na camada de apresentação.

É uma *framework* concebida em torno de um *DispatcherServlet* que distribui os pedidos para *handlers* onde se faz o tratamento de ações. Na próxima imagem é demonstrada esta arquitetura.

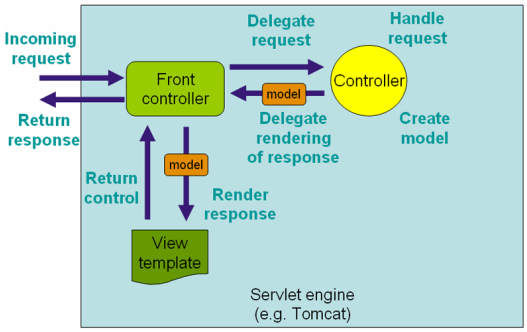

Figura 4: Arquitetura da Framework Spring MVC

Relativamente aos aspectos, que anteriormente mencionamos, que servem de base para a avaliação da *framework*, concluí-se o seguinte:

Prototipagem rápida da aplicação Se estamos a procura de uma *framework* que ajude a criar uma aplicação rapidamente, o Spring MVC não é o ideal. Esta *framework* é enorme o que dificulta, inicialmente, a percepção da mesma.

Complexidade Spring MVC tem uma base sólida, construída ao longo dos anos, que aumenta o âmbito da sua aplicação e consequentemnete a sua complexidade.

A arquitetura da mesma é relativamente simples, mas há ainda muitas camadas e abstrações que dificultam o *debug* se algum problema acontecer.

Facilidade de utilização Spring MVC não é muito fácil de utilizar, devido à grandeza e complexidade da mesma. Assim para ser utilizador desta *framework* é necessário já ter alguns conhecimentos da arquitetura.

Documentação e Comunidade Como existem milhares de *developers* a trabalhar com esta *framework* existe muita documentação. Até o próprio site contém inúmeros tutoriais, o que é útil para iniciantes, e mesmo utilizadores avançados.

A comunidade também é muito forte, tanto a nível de suporte a dúvidas de utilizadores, como a correções de funcionalidades da *framework*.

Ecossistema O ecossistema do Spring MVC é bem desenvolvida. Beneficia de ferramentas como o Spring Roo e o Spring Tool Suite IDE.

Escalabilidade Spring MVC é utilizado em aplicações de grande escala em todo o mundo. Contém os componentes necessários para o processamento em paralelo e construção de aplicações *multi-thread*.

Pode ser dividida em diferentes módulos e ser configurado, mais tarde, em diferentes hosts.

Manutenção e Atualizações A manutenção de código é complicada se não forem conhecidos os pormenores da *framework*.

User Experience Tem um rico conjunto de recursos para desenvolver e manter o código no servidor, mas apesar da sua extensibilidade não fornece qualquer construção de interface.

Os módulos são fáceis de gerir e criar.

Assim verifica-se que a *framework* apresenta excelentes recursos de escalabilidade, ecossistema, documentação e ainda um forte apoio da comunidade. Para aplicações de baixa complexidade ou prototipagem rápida a escolha desta *framework* não é a mais indicada.

4.2 Grails

É uma *framework* para a construção de aplicações web através da linguagem dinâmica *Groovy* para a plataforma Java. Grails é uma *framework open-source* e pretender dar alta produtividade graças à utilização do paradigma de mais convenção e menos detalhes de configuração.

Utiliza as tecnologias que são consideradas as mais maduras do mundo de Java, como as *frameworks Hibernate* e *Spring*, através um interface simples e consistente.

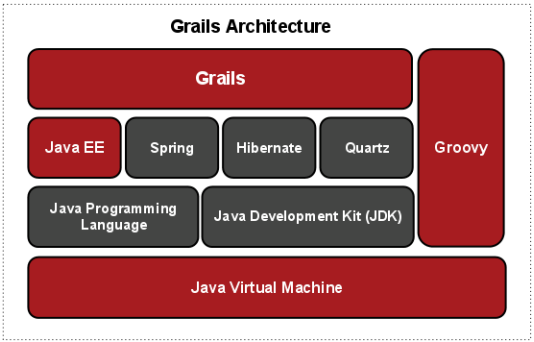

Figura 5: Arquitetura da Framework Grails

Prototipagem rápida da aplicação Grails oferece uma prototipagem simples, facilitando a instalação. Possui mecanismos para geração de código automática, diminuindo o tempo de desenvolvimento.

É muito interessante utilizar esta *framework* para projetos de pequena-média dimensão.

Complexidade Grails está rodeado por *frameworks* de outras tecnologias como o Spring MVC e o Hibernate. Tem a mesma complexidade que estas *frameworks*, acrescentado ainda uma outra camada de abstração.

Facilidade de utilização A *framework* foi projetada para ser de desenvolvimento rápido e direta facilidade de utilização. Permite o uso de *plugins*, através de um comando na consola todas as depedências e configurações são geradas.

Documentação e Comunidade Tem uma comunidade enorme e dedicada. Possui muita e variada documentação, inclusive, *screencasts*.

Ecossistema Grails é uma *framework full-stack*, onde muitas das peças fornecidas podem ser substituídas. Existe uma grande variadade de *plugins* disponiveís na sua biblioteca.

Escalabilidade A *framework* é uma abstração sobre Spring e Hibernate, ambos escaláveis, e sem influência no rendimento.

Manutenção e Atualizações A manutenção é fácil por não haver muita configuração e leva a que as estruturas dos projetos sejam muito semelhantes.

User Experience Os *plugins* fornecidos tem fácil intregração com JavaScript e possui muitos componentes de interface.

Podemos concluir que esta *framework* apresenta características impressionantes, que facilitam a prototipagem rápida e o desenvolvimento ágil das aplicações. Como referido anteriormente Grails é ótimo para projetos de média-baixa escala.

4.3 GWT - Google Web Toolkit

É um *toolkit open-source* que permite aos *developers* criar aplicações com tecnologia AJAX em linguagem de Programação Java.

Prototipagem rápida da aplicação Possui uma quantidade enorme de *widgets* para uso rápido e permite com o uso de JavaScript fazer uso de qualquer coisa.

O GWT Desgin Mode, permite através de drag-and-drop fazer uma interface fácil com geração automática de código.

Complexidade É bastante complexo. Devendo-se à enorme quantidade de código que pode ser gerada automaticamente.

Facilidade de utilização GWT possui uma estrutura bastante simples para programadores e designers. O código é fácil de interpretar e escrever.

Documentação e Comunidade Tem uma surpreendente extensa documentação oficial. O site do projeto possuí tutoriais e documentação detalhada, desde a parte mais simples à parte mais complexa.

A Google estabeleceu um conjunto de grupos de discussão que permite aos utilizadores obter suporte em questões técnicas.

Ecossistema GWT permite construir aplicações rapidamente mantendo o JavaScript do front-end com elevada perfomance em Java. O GWT SDK permite que se escreva as aplicações AJAX em java e depois compila-se o código em JavaScript.

Escalabilidade GWT foi criado para escalabilidade. O compilador Java converte JavaScript com a máxima eficiência e escalabilidade.

Manutenção e Atualizações Muito fácil de manter atualizada, mas o código gerado automaticamente pode ser díficil de interpretar.

User Experience GWT é muito versátil quando se trata de personalizar a aparência das aplicações web. É uma *framework* baseada em componentes, onde a qualquer momento se pode modificar o layout sem haver uma ruptura no sistema.

É possível concluir que GWT é uma boa *framework*, mas como se vai poder analisar mais há frente, dentro desta categoria já é possível encontrar *frameworks* melhores.

4.4 Play!

Play! é uma *framework* que torna a construção de uma aplicação web mais facilitada em Java ou Scala. É uma arquitetura muito leve e *web-friendly* consumindo os recursos mínimos para aplicações altamente escaláveis.

Passando à análise das características da *framework*, verifica-se:

Prototipagem rápida da aplicação Play! é muito simples. Um dos objetivos de Play! é ter a mesma facilidade de criação de aplicações web que tem as aplicações em Ruby On Rails.

Complexidade Tem uma estrutura muito complexa pois existe uma vários mecanismos e módulos que podem ser movidos. É necessário ter algumas noções de Scala, caso contrário, a complexidade aumenta.

Facilidade de utilização É muito simples de começar, no entanto, fica mais complexo para os programadores mais avançados.

Documentação e Comunidade Tem uma comunidade de tamanho razoável com excelente documentação interna e externa. Existe ainda vários tutoriais a explicar as características de *scaffolding*.

Ecossistema Play! oferece tudo o que é necessário para desenvolver, executar e testar as aplicações. A desvantagem real é a dependência de Scala.

Escalabilidade A *framework* é incrivelmente escalável quando combinada com *Acca* permitindo alta taxa de transferência.

Manutenção e Atualizações O código produzido pela *framework* é legível e bem estruturado. Permite que novos utilizadores começem a trabalhar na aplicação sem perder demasiado tempo a perceber o funcionamento da mesma.

User Experience Possuí alguns temas básicos, no entanto não há um apoio generalizado para bibliotecas de componentes.

4.5 JSF - JavaServer Faces

JavaServer Faces (JSF) é uma framework MVC baseada em Java para a construção de interfaces baseadas em componentes para aplicações web. Tem um modelo de programação direcionado a eventos, abstraindo os detalhes da manipulação dos eventos e organização dos componentes.

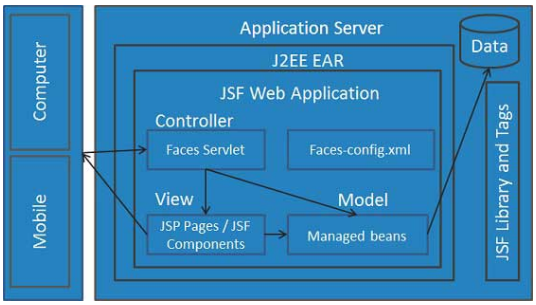

Figura 6: Arquitetura da Framework JSF

Obteve-se as próximas conclusões, relativas às características definidas inicialmente.

Prototipagem rápida da aplicação JSF não é a melhor para a elaboração rápida de protótipos pois exige tanto de configuração como uma aplicação completa.

Os maiores ganhos na produtividade com JSF são os assistentes disponíveis na maioria dos IDEs que geram a maior parte do código.

Complexidade É muito complexa, derivado à complexidade da especificação JAVA EE.

Facilidade de utilização JSF tenta proporcionar uma estrutura fácil de utilização para a criar componentes web reutilizáveis nas aplicações Java.

As ferramentas da *framework* tornam mais fácil a utilização e não possui depedências externas enquanto estivermos dentro do ecossistema Java EE, que JSF aproveita bem.

Documentação e Comunidade JSF é totalmente suportado pela Oracle. Ao contrário de outras *frameworks* JSF tem funcionários para escrever documentação e criar amostras e exemplos.

A única desvantagem da documentação da Oracle é a dependência do seu IDE e Application Server.

Ecossistema Porque é parte da especificação de Java EE, JSF recebe todos os benefícios e ferramentas Java EE. Incluíndo suporte para IDEs com muitas funcionalidades.

Escalabilidade As aplicações JSF podem ser realizadas, mas os ganhos de desempenho vêm do cluster de servidores de aplicações Java EE. O próprio JSF não fornece suporte explícito para chamdas assíncronas.

Manutenção e Atualizações Um dos objetivos de JSF é ajudar os programadores a produzir um código melhor.

User Experience As bibliotecas de construção de componentes está cheia de recursos e há um extenso catálogo *3rd-party* disponíveis.

4.6 Outras frameworks

Derivado à grande quantidade de *frameworks* existentes para a camada de apresentação não foi possível apresentar todas as características e explicações das mesmas.

Merecem ainda algum destaque as seguintes *frameworks*:

Struts É uma *framework* baseada em ações que segue o padrão Model 2, que é uma variação complexa do padrão MVC.

Wicket Desenvolvida pela Apache FOundation, é semelhante ao JavaServer Faces (JSF). Caracteriza-se por centralizar em POJO (Plain Old Java Objects) e evitar o excesso de ficheiros de configuração em XML.

Vaadin É uma *framework* desenvolvida a partir da *framework* GWT - Google Web Toolkit. Esta opera no lado do servidor, embora também funcione no lado do cliente, recorrendo a tecnologia AJAX.

4.7 Decisão Final

Depois de feita a comparação entre as várias características existentes numa *framework* foi feita a escolha de uma, que neste caso completa mais os tópicos analisados.

Para nos ajudar nesta escolha recorremos ao artigo da RebelLabs que analisou cuidadosamente as características de várias *frameworks* e atribuíu pontuações às mesmas.

	Rapid Application Development	Framework Complexity	Ease of Use	Documentation & Community	Framework Ecosystem	Throughput/ Scalability	Code Maintenance/ Updates	UX/Look and Feel
Spring MVC	2.5	3.5	3	4	4	4	3	2
Grails	**5**	3	**4.5**	**5**	**4.5**	4	**4.5**	4
Vaadin	4.5	**4**	**4.5**	**5**	3	**4.5**	4	**5**
GWT	4	**4**	4	4.5	3	**4.5**	4	**5**
Wicket	3.5	2.5	3.5	3	3	3	**4.5**	3.5
Play	**5**	2	3.5	4	**4.5**	5	4	3
Struts	2	**4**	3	2.5	3	3	3	2.5
JSF	3	3.5	4	4.5	4	4	4	4.5

Figura 7: Análise das frameworks

Com a seguintes pontuações totais ordenadas:

Framework	Overall Score
Grails	**34.5**
Vaadin	**34.5**
GWT	33
JSF	31.5
Play	31
Wicket	26.5
Spring MVC	26
Struts	23

Figura 8: Pontuações obtidas por Framework

Estas tabelas refletem que as *frameworks Grails e Vaadin*, apresentam-se como as mais completas nos tópicos analisados anteriormente.

A *framework Grails*, através da análise das tabelas anteriores mostrou que para as 8 categorias analisas ela é bastante completa em 5. Particularmente em desenvolvimento rápido de aplicações, documentação e comunidade, com as melhores notas. O tópico que ficou mais abaixo dos valores foi nas comparações de complexidade, este valor reflete as depedências com Spring e Groovy, mais a adição de camadas de abstração.

A *framework Vaadin*, como verificado nas tabelas anteriores esteve muito bem em 3 categorias. Apesar de não termos efetuado uma comparação exaustiva da mesma, serve para demonstrar que é uma boa *framework* de desenvolvimento. Esta saiu mais destacada na análise de User Experience e também na documentação e comunidade.

Estas duas *frameworks* são uma excelente opção para projetos de pequena-média dimensão, no entanto também se enquadram em grandes dimensões devido à sua escalabilidade e desempenho.

Concluí-mos então que as duas frameworks faladas em cima são as melhores para a análise efetuada.

5 Conclusão

No presente relatório foram analisadas algumas das várias frameworks em Java presentes em cada uma das três camadas, tentado ter um suporte para a escolha da melhor quando for necessário. Para a escolha de uma framework é necessário analisar o contexto em que esta irá estar inserida, e o tipo de projeto que se pretende construir.

A separação de camadas permite tornar os sistemas mais flexivéis, de modo a que se altere cada camada sem necessidade de se alterarem as outras. As camadas analisadas anteriormente são as principais, mas é ainda possível dividir estas camadas em subcamadas.

Com a elaboração deste trabalho, conseguímos perceber que não se deve optar por uma framework por ser uma tendência do momento, mas sim refletir sobre qual, para o projeto pretendido irá tirar mais partidos das características descritas ao longo do relatório dentro da própria infraestrutura computacional.

6 Referências

Camada de Apresentação