UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA

Arquiteturas Aplicacionais

Frameworks para desenvolvimento de aplicações web multi-camada

Ana Marta Santos Ribeiro A82474
Jéssica Andreia Fernandes Lemos A82061
Miguel José Dias Pereira A78912

MIEI - 4º Ano - Arquiteturas Aplicacionais

Braga, 19 de Março de 2020

Conteúdo

1 Introdução

Este relatório aborda algumas das *frameworks* existentes no mercado para o desenvolvimento de aplicações web. Esta análise foi realizada considerando a existência de três camadas, nomeadamente a de dados, a de negócio e a de apresentação. Desta forma, para cada uma das camadas foi feito um estudo sobre as *frameworks* que consideramos mais adequadas. A seleção das ferramentas a analisar teve por base o grau de utilização das mesmas.

Finalmente para cada uma das camadas selecionamos a *framework* que consideramos mais pertinente de forma a elaborar uma arquitetura final que seja o mais indicada possível para o desenvolvimento de uma aplicação web.

2 Frameworks

2.1 Camada de Dados

Esta camada é reponsável por garantir a persistência dos dados relativos aos objetos na Base de Dados. Tal pode ser realizado através de uma abordagem centrada tanto nos objetos como na Base de Dados. Assim sendo, nesta secção vamos abordar diferentes *frameworks* com diferentes características a título de exemplo.

2.1.1 Hibernate

O *Hibernate* é uma *framework open source* que permite realizar o mapeamento entre as tabelas da base de dados e as classes definidas pela aplicação. Basicamente funciona como uma camada intermédia entre ambas. Assim as classes JAVA podem ser transformadas em tabelas da base de dados e os tipos dos seus objetos em tipo compativeis *SQL*. Tal pode ser realizado através de arquivos *XML* ou de anotações no código fonte.

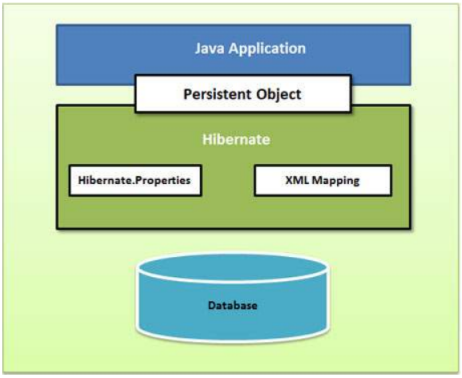

Figura 1. Arquitetura da framework *Hibernate*

Uma das suas principais características é a existência de uma linguagem própria, nomeadamente *HQL*(Hibernate Query Language) totalmente orientada a objetos. Assim é possível utilizar tanto a linguagem *SQL* como a *HQL* o que se revela uma mais valia.

2.1.2 MyBatis

A *MyBatis* é também uma *framework open source*, no entanto, constitui uma alternativa às *frameworks ORM* como é o caso da *Hibernate*. Basicamente, a aplicação é

desenvolvida centrada na base de dados e não nos objetos. Como não possui a sua própria linguagem esta utiliza recorrentemente *SQL*, revelando-se dependente da Base de Dados o que não é muito positivo. A configuração para o mapeamento dos objetos JAVA para a base de dados pode ser realizada através de ficheiros *XML* ou de anotações.

2.1.3 jOOQ

Tal como a *MyBatis*, esta *framework* também se apresenta como uma alternativa às *frameworks ORM*, sendo centralizada na Base de Dados. Assim sendo, esta permite gerar código JAVA a partir da Base de Dados e ainda escrever código JAVA com uma sintaxe similar à do *SQL*. Uma das suas principais características são os active records que facilitam as operações CRUD. A standardização é também um aspeto essencial, dado que permite adaptar as expressões automaticamente à Base de Dados, não sendo assim dependente do *SQL* inerente. É importante referir que suporta os *procedures*. Uma das suas desvantagens relativamente à *MyBatis* é o facto de ser necessário uma licença para o uso.

2.2 Camada de Negócio

Esta trata-se da camada intermédia na qual encontra-se a lógica aplicacional desenvolvida para o software. É também responsável por interligar as duas outras camadas. Desta forma, pretendemos investigar *frameworks* que facilitem a implementação ágil e simples de componentes modulares. De seguida, apresentamos algumas das ferramentas relevantes para esta camada.

2.2.1 Spring

A ferramenta trata-se de uma *framework open-source* full-stack que tem como objetivo tornar o desenvolvimento de aplicações mais simples, sendo uma das principais características o facto de não precisar de um servidor de aplicação para funcionar. Esta possuí ferramentas aplicáveis em inúmeros contextos, desde segurança a web services, e para além disso, apresenta diversos módulos, apresentados na **Figura 2**, que podem ser utilizados tendo em conta o que é pretendido.

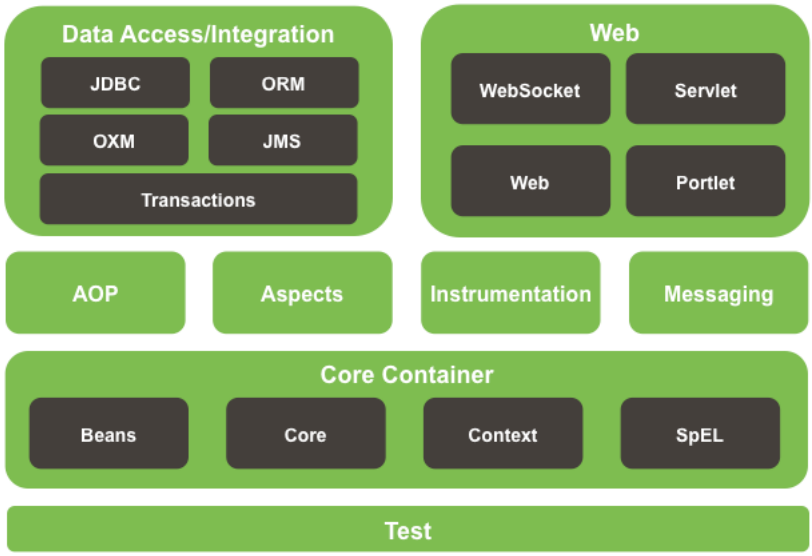

Figura 2. Estrutura da framework *Spring*

A *Spring* foi desenvolvida para *Java*, pelo que tem por base conceitos como:

- **Inversão de controle (IoC) -** corresponde à transferência do controlo de objetos para um container ou framework, permitindo separar o serviço da sua implementação
- **Injeção de dependência (DI) -** permite combater a definição estática dos valores associados a variáveis de instâncias de uma classe

Como tal, fornece um container que representa o núcleo da framework e é responsável por criar e gerir os componentes da aplicação, os *beans*.

Esta é uma das *frameworks open source* mais antigas e populares, tendo uma notória comunidade que a permite estender, mantendo-a sempre atualizada e moderna. Desta forma, temos uma ferramenta compatível com várias outras frameworks.

2.2.2 Google Guice

Esta framework *open source lightweight* de injeção de dependências, habitualmente em componentes *server-side*, assim como a *Spring* contudo bastante mais simples sendo que não existe a necessidade de configurar através de vários ficheiros XML. Esta recorre a anotações para configurar os objetos. Basicamente, a *Google Guice* promove e facilita a execução de testes, em parte devido à utilização de DI, sendo possível efetuar testes aos *bindings* e *modules* bem como a alteração e reutilização do código. Além

disso, possuí um sistema de erros claro. Desta forma, este é uma *framework* de simples utilização.

2.2.3 CSLA

A CSLA .NET é uma *framework* que ajuda na criação de uma camada de negócios orientada a objetos reutilizáveis e sustentáveis para a aplicação. Desta forma, os objetos CSLA ganham automaticamente muitos recursos avançados que simplificam a criação de interfaces. Esta permite aproveitar o poder do design orientado a objetos para reduzir o custo de criação e manutenção.

Esta *framework* permite ainda grande flexibilidade na persistência de objetos, sendo possível os objetos de negócio usar praticamente qualquer fonte de dados disponível. A estrutura suporta modelos de 1, 2 e n camadas devido ao conceito de objetos móveis. Assim, é possível otimizar o desempenho, escalabilidade, segurança e tolerância a falhas sem a alteração de código na interface ou nos objetos de negócio.

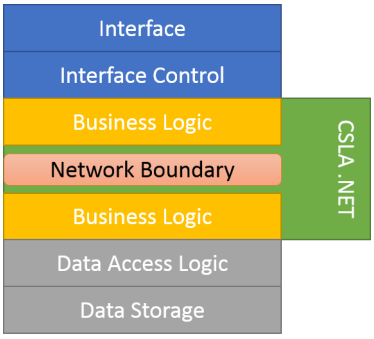

Figura 3. Estrutura da framework *CSLA*

2.3 Camada de Apresentação

A camada de apresentação é o*front-end* da aplicação e consiste na interface do utilizador. Nesse sentido fizemos um levantamento de *frameworks* que permitissem ajudar no desenvolvimento da interface e no *deploy* da mesma, simplificando o processo de codificação.

2.3.1 JSF

A *JavaServer Faces* é uma *framework* desenvolvida pela Oracle e permite criar interfaces para aplicações *web* baseadas em Java. Esta *framework* é baseada no padrão

MVC e apresenta uma arquitetura que define uma separação bem distinta entre as camadas de negócio e apresentação, tudo numa lógica *server-side* orientada a eventos. Para além disso, a *framework* JSF é *component-based*, permitindo assim utilizar componentes pré-definidos e reutilizáveis.

Para a criação das interfaces o sistema de *templates* mais utilizado é o *Facelets*. Cada vista é contruída como uma página XHTML, combinando componentes compostos e *tags*, entre outros, definindo assim interfaces JSF.

Um dos aspetos chave desta *framework* é permitir encapsular tecnologias *client-side* como *HTML*, *CSS* e *JavaScript*, dando assim possibilidade aos desenvolvedores de construir interfaces *web* sem ser necessário ter um conhecimento aprofundado sobre estas tecnologias.

Esta *framework* tem algumas desvantagens como a complexidade e dificuldade de aprendizagem da *framework*.

2.3.2 GWT

O *Google Web Toolkit* é uma *framework open-source* que permite desenvolver interfaces web utilizando *Java*, e permite fazer o balanceamento de carga entre cliente e servidor.

Uma das características chave desta *framework* é o facto de ser um compilador de *Java* para *JavaScript*, *HTML* e *CSS*, o que permite escrever toda a aplicação *web* em *Java*, tirando assim partido de todas as vantagens que o *Java* apresenta. Por outro lado o compilador cria código otimizado criando assim aplicações de alta performance.

Outras características interessantes do *GWT* são os componentes reutilizáveis e dinâmicos, tratar da compatibilidade dos *browsers*, a aprendizagem da *framework* ser relativamente fácil e o *debug* poder ser feito a partir de um *IDE Java*.

Por outro lado também apresenta aspetos negativos como o facto de ser um compilador o que torna mais complicado a manipulação e personalização do *JavaScript*, *HTML* e *CSS*, e também não tem componentes pré-definidos.

2.3.3 Vaadin

Vaadin é uma *framework open-source* para desenvolver a camada de apresentação de aplicações *web* em *Java*. Suporta ambas as arquiteturas *client-side* e *server-side*, mas este último é o mais utilizado e o mais poderoso.

Esta *framework* permite criar aplicações *web* complexas e dinâmicas com grande facilidade para o desenvolvedor totalmente escritas em *Java*, o que permite não ter um conhecimento aprofundado sobre *JavaScript*, *HTML* e *CSS*. Pode ser utilizada em conjunto com o *Ajax* para criar aplicações que correm no browser para garantir interatividade e melhorar a experiência do utilizador.

Sendo que o foco do *Vaadin* está no desenvolvimento de *user interfaces*, esta *framework* disponibiliza múltiplos componentes pré-definidos com foco na performance e experiência do utilizador.

No entanto como o *HTML* e *CSS* são gerados automaticamente torna-se difícil personalizar as interfaces, e o código gerado por vezes torna-se extenso.

2.3.4 ASP.NET

ASP.NET é uma *framework open-source server-side* desenvolvida pela *Microsoft* desenhada para produzir páginas web dinâmicas, que estende a *framework .NET* com componentes para a construção de aplicações web. Existem outras versões como *ASP.NET MVC*, que utiliza o padrão *MVC* ao invés de *web forms* e *postbacks* utilizados pelo *ASP.NET*, e ainda a *ASP.NET Core*, que é uma nova versão do *ASP.NET* com a vantagem de correr nas plataformas *Windows*, *MacOS* e *Linux*, enquanto que a *ASP.NET* só corre em *Windows*.

A *framework ASP.NET* tem como vantagens não só ser *open-source*, mas também permitir criar aplicações rápidas e escaláveis, mas por outro lado a documentação não é a melhor e só opera em *Windows*.

3 Arquitetura Proposta

Estudadas as ferramentas para cada camada de uma aplicação, pretendemos agora propor uma arquitetura a ter em conta no desenvolvimento do trabalho prático.

3.1 Camada de Dados

Tendo em conta as *frameworks* apresentadas anteriormente referentes à camada de dados, optamos por escolher para a nossa arquitetura a *Hibernate*. Apesar de apresentar alguns aspetos menos positivos, consideramos que o facto de esta possuir uma linguagem própria é fundamental, visto que permite que o código seja independente da Base de Dados. Para além disso é uma das mais utilizadas no mercado o que evidencia a sua qualidade e possui uma extensa documentação, o que permitirá resolver mais facilmente os obstáculos encontrados durante a implementação da aplicação. É importante realçar ainda que esta framework é compatível com *Spring*, que foi a framework selecionada para a camada de negócio.

3.2 Camada de Negócio

Para esta camada optamos por utilizar a *framework Spring*. Esta escolha deve-se a que esta ferramenta fornece bastante liberdade nas configurações, embora não sejam tão compactas e simples comparativamente à *Guice* e as capacidades da primeira também são mais extensas. Além disso, sendo a adesão a esta bastante grande torna-se possível acrescentar novos módulos sem problemas associados. Outro aspeto que também tivemos em consideração foi que a conexão à *framework Hibernate*, ferramenta escolhida para a camada anterior, é simples.

Desta forma, esta foi a ferramenta mais completa que estudamos para esta camada.

3.3 Camada de Apresentação

Como pretendemos usar Java para o desenvolvimento temos apenas 3 opções *JSF*, *GWT* e *Vaadin*. Visto que a *JSF* é uma *framework* complexa e de difícil aprendizagem, esta foi descartada.

Ficamos então com a *GWT* e *Vaadin*, que são bastante parecidas em termos de vantagens e desvantagens, e a escolha recaiu sobre a *GWT*, pois esta *framework* é

relativamente simples de aprender, o código gerado é otimizado e de alta performance e o *debug* pode ser feito de maneira simples através do *IDE*.

Para além disso, a *GWT* também traz outras vantagens como o balanceamento de carga que permite fazer entre cliente e servidor e é compatível com a *framework* escolhida para a camada de negócio *Spring*.

3.4 Desenho da Arquitetura

Dado as *frameworks* que selecionamos para as várias camadas desenvolvemos o seguinte diagrama que representa a nossa proposta de arquitetura.

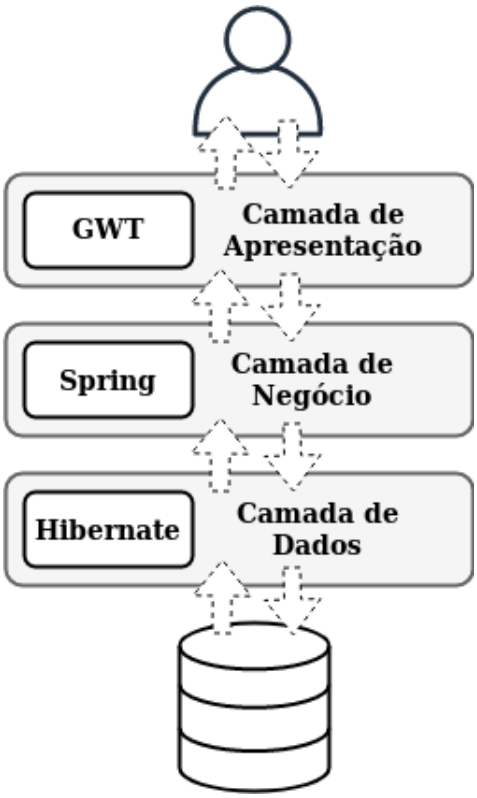

Figura 4. Arquitetura Proposta

4 Conclusão

Com este trabalho tornou-se possível explorar as várias *frameworks* existentes no mercado, bem como perceber as que melhor se apropriavam para cada uma das camadas de uma aplicação.

Tendo em conta que 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 através do estudo de algumas destas ferramentas, conseguimos adquirir sensibilidade e conhecimento para selecionar as que se apropriam melhor a cada contexto, e como tal propor uma arquitetura para o projeto prático que iremos desenvolver. Contudo, esta poderá sofrer alterações na fase de implementação.

Em última instância, conseguímos percecionar que a escolha uma *framework* deve ter como motivação tirar partido das suas características dentro da infraestrutura computacional.

Bibliografia

[1] https://www.javatpoint.com/hibernate-tutorial

[2] https://www.tutorialspoint.com/hibernate/hibernate_quick_guide.htm

[3] https://www.marcobehler.com/guides/jooq

[4] https://www.tutorialspoint.com/mybatis/mybatis_overview.htm

[5] https://www.tutorialspoint.com/spring/spring_overview.htm

[6] https://docs.spring.io/spring/docs/4.3.x/spring-framework-reference/html/overview.html

[7] https://www.tutorialspoint.com/guice/index.htm

[8] https://cslanet.com/

[9] https://www.dailyrazor.com/blog/best-java-web-frameworks/

[10] https://hackr.io/blog/java-frameworks

[11] https://raygun.com/blog/popular-java-frameworks/

[12] https://www.javaworld.com/article/3322533/what-is-jsf-introducing-javaserver-fa
html

[13] https://docs.oracle.com/javaee/6/tutorial/doc/gijtu.html

[14] https://medium.com/@alextheedom/what-is-javaserver-faces-jsf-19c20da594f4

[15] https://hackr.io/blog/java-frameworks

[16] https://www.toptal.com/front-end/javascript-front-ends-in-java-with-gwt

[17] http://www.gwtproject.org/overview.html

[18] https://www.wakefly.com/blog/what-is-asp-net-and-why-should-i-use-it/