Universidade do Minho

Departamento de Informática

Mestrado Integrado em Engenharia Informática

Engenharia de Aplicações

Arquiteturas Aplicacionais

Frameworks

Carlos Pinto Pedrosa A77320
David José Teixeira de Sousa A78938
Isabel Sofia da Costa Pereira A76550
Maria de La Salete Dias Teixeira A75281

25 de Fevereiro de 2019

Conteúdo

1 Introdução

Uma *framework* é definida como um uma abstração que une partes de código comuns entre vários projetos de *software*, podendo estar direcionada a uma funcionalidade específica de uma aplicação ou ser uma ferramenta mais genérica. Assim, cada vez mais os engenheiros de *software* recorrem a *frameworks* para facilitar o desenvolvimento de aplicações web. Dependendo da *framework* em causa, esta pode ser capaz de auxiliar o desenvolvimento de todas as camadas da aplicação web, como pode ser específica para o desenvolvimento de uma ou mais camadas.

Considerando as três camadas de uma aplicação como a camada de dados, camada de negócio e camada de apresentação, neste relatório vamos identificar e comparar as *frameworks* que consideramos ser mais adequadas para cada uma destas para, no final, optar por uma *framework* considerada ideal para cada uma das camadas.

Na seleção das *frameworks* a estudar, optou-se por apenas analisar para *Java* e *.NET*. Sendo que não seria possível estudar todas as *frameworks* existentes, teve-se o cuidado de selecionar as mais respeitadas e mais utilizadas entre as empresas de *software*.

2 Conceitos Base

Com a utilização de *frameworks*, surgem alguns conceitos que se consideram relevantes apresentar, para que o seu uso no seguimento do relatório seja claro.

- ***Server-side***: aplicações que recebem pedidos *HTTP* (por exemplo, um cliente entrar numa página web), levando à execução de código por parte do servidor web, gerando um resultado em *HTML* que é enviado como resposta ao cliente. Desta forma, a página do utilizador só é modificada quando o servidor receber um novo pedido, não sendo então possível manipular a página do cliente em tempo real.

- ***Client-side***: aplicações em que o código é transferido desde o servidor web até ao computador do cliente, via *internet*. Posteriormente, esse código é executado na máquina do cliente, gerando uma página web. Como o código encontra-se do lado do cliente, o utilizador é capaz de, por exemplo, alterar um valor num campo de um formulário, sendo que o código consegue reagir aos *inputs* do utilizador. Além disso, é possível que o cliente veja o código e manipule-o, gerando possíveis questões de segurança.

- ***Mixed solutions***: aplicações que utilizam simultaneamente soluções *client-side* e *server-side*, levando a que haja código a ser executado dos dois lados. Isto faz com que seja possível apresentar interfaces mais interativas e fáceis de utilizar, ao mesmo tempo.

- ***MVC Pattern***: o *Model-View-Controller* é composto por três *design patterns* do *Java*. O *Model* utiliza o *design pattern Observer* para manter as *views* e os *controllers* atualizados. A *View* utiliza os *design patterns Strategy* e *Composite*, sendo este último utilizado para gerir os componentes das páginas. E, por último, o *Controller* utiliza o *design pattern Strategy*.

3 *Frameworks*

3.1 *Frameworks* para Camada de Dados

Toda a lógica envolvida na configuração da ligação da Base de Dados ao *software* pode ser extremamente custosa em termos de tempo e de dificuldade. Assim, neste capítulo, irá-se identificar e descrever algumas *frameworks* capazes de minimizar este problema.

3.1.1 *Hibernate*

A *Hibernate* é uma *framework open-source* capaz de mapear modelos orientado a objetos numa base de dados relacional. O *Hibernate* atua como uma camada intermediária entre a aplicação e a base de dados e que transforma as instruções *HQL* em *SQL* para que possa ser interpretada pela Base de Dados. Uma das principais características desta ferramenta é o mapeamento de clases *Java* para tabelas da Base de Dados e o de tipos de dados *Java* para tipos de dados *SQL*. Para além disto, o *Hibernate* também facilita as interrogações à Base de Dados e a manipulação dos resultados dessas *queries*, gerando o *SQL* necessário e facilitando o trabalho do programador.

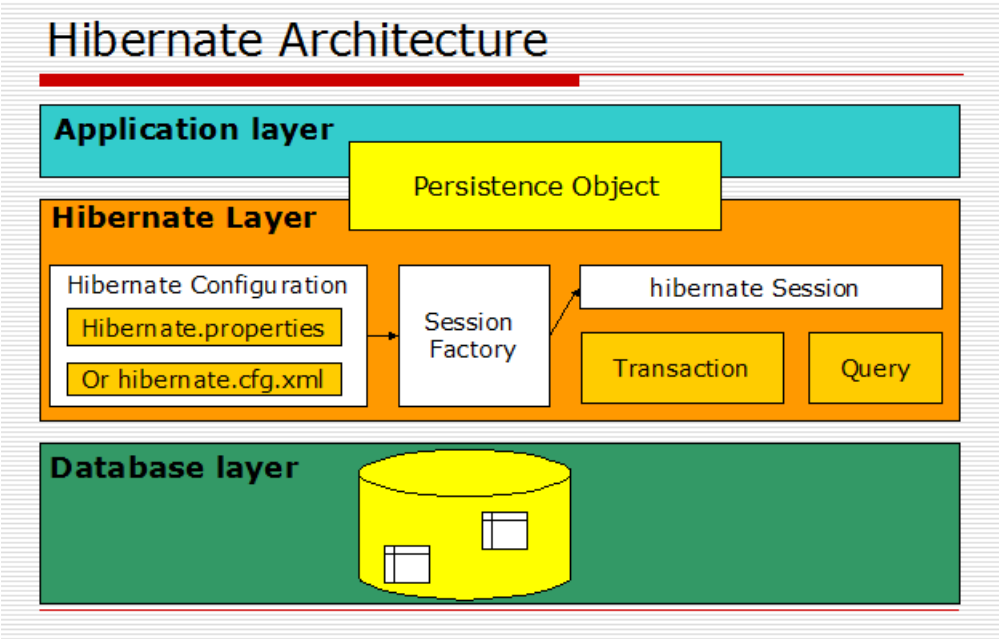

Figura 1: Arquitetura da *Hibernate*

Vantagens

- É uma ferramenta *open-source*;
- Alto Desempenho, Escalabilidade, Confiabilidade e Extensibilidade;
- Para além de uma *API* nativa, o *Hibernate* é também uma implementação da *JPA* (*Java Persistence API*), o que permite integração com qualquer ambiente que suporte *JPA*, como aplicações *Java EE*, servidores aplicacionais *Java SE*, *Enterprise OSGi containers*, entre muitos outros;
- Suporta herança, associações e coleções;
- Suporta todos os tipos de relações;
- Capacidade de gerar chaves primárias automaticamente;
- Tem a sua própria linguagem, *Hibernate Query Language(HQL)* o que permite mudar o Motor de Base de Dados sem alterar qualquer código;

- Menor repetição de código quando comparado com *JDBC*;
- Existe muita documentação *online* para além da original;

Desvantagens

- Custos de Desempenho devia do *SQL* gerado em *runtime*;
- Uso de *stored procedures* é relativamente complicada;
- Não permite múltiplos *Inserts*;
- Em projetos de pequena dimensão introduz mais *overhead* do que benefícios;
- Curva de Aprendizagem.

3.1.2 *MyBatis*

A *MyBatis* é uma *framework* de persistência de dados com suporte a *SQL* modificado, *stored procedures* e mapeamento avançado. Esta ferramenta elimina quase todo o código *JDBC*, configuração manual e manipulação de resultados. Esta usa *XML* ou *Annotations* para configuração e mapeamento de objectos Java para a Base de Dados.

Vantagens

- Ferramenta *open-source*;
- Simplicidade;
- Rápido Desenvolvimento de Aplicações;
- Portabilidade: Pode ser usado em *Java*, *Ruby* e *.NET*;
- Suporta Procedimentos, *inline SQL* e *SQL* dinâmico;
- Suporta quase todas as características *ORM (Object-Relacional Mapping)*;

Desvantagens

- Não tem a sua própria linguagem, o que faz com que seja dependente da Base de Dados;
- Não pode ser usado com Base de Dados não Relacionais;

MyBatis	Hibernate
It is simpler. It comes in a much smaller package size.	Hibernate generates SQL for you, which means you don't have to spend time on generating SQL.
It is flexible, offers faster development time.	It is highly scalable, provides a much more advanced cache.
It uses SQL, which could be database dependent.	It uses HQL, which is relatively independent of databases. It is easier to change db into Hibernate.
It maps the ResultSet from JDBC API to your POJO Objects, so you don't have to care about table structures.	Hibernate maps your Java POJO objects to the Database tables.
It is quite easy to use stored procedure in MyBatis.	Use of stored procedures is a little difficult in Hibernate.

Figura 2: Comparação entre *Hibernate* e *MyBatis*

Vale notar que ambas as ferramentas são compatíveis com *Spring*.

3.1.3 *Entity Framework*

A *Entity Framework* é uma ferramenta orientada para *.NET* que atua como mapeador entre objetos e modelos relacionais (*ORM*) e que permite aos programadores interagir com a Base de Dados através de objetos *.NET* eliminando assim quase todo o código necessário para aceder aos dados na Base de Dados.

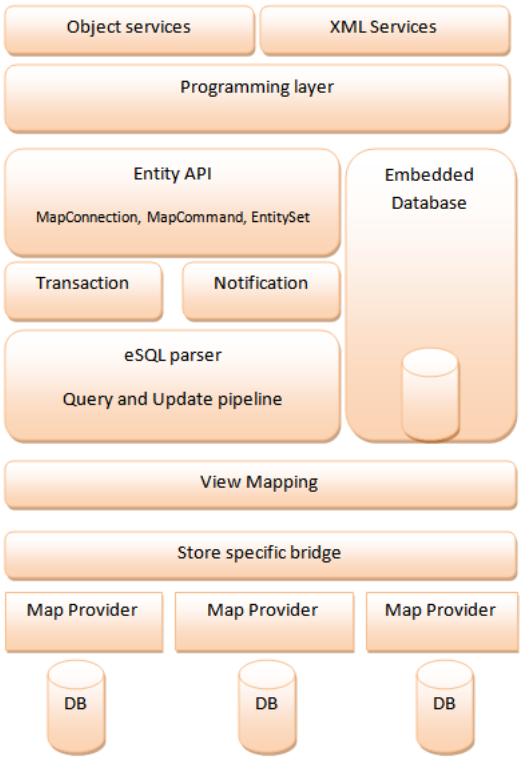

Figura 3: Arquitetura da *Entity Framework*

Vantagens

- Geração automática de código para aceder à Base de Dados;
- Redução do tempo e respetivos custos de desenvolvimento;
- Sintaxe *LINQ* para todas as *queries*;
- Simplificação geral das *queries*;
- Abstração do modelo de dados;
- *Code-First*: a *Entity Framework* permite criar uma Base de Dados a partir de um modelo ou criar um modelo de dados a partir de uma Base de Dados já existente.

Desvantagens

- Alterações na Base de Dados provocam uma alteração na EF;
- Não está disponível para todos os RDMS;
- SQL Dinâmico pode ser um fator de quebra de desempenho;
- Não escalável para grande domínios;
- Não pode ser usado com Base de Dados não Relacionais;

3.2 *Frameworks* para Camada de Negócio

O principal objetivo na utilização de *frameworks* para a camada de negócio é a simplificação do código a implementar. Desta forma, apresentam-se de seguida algumas *frameworks* relevantes para esta camada.

3.2.1 *Spring*

A *Spring* trata-se de uma *framework open source*, criada por Rod Johnson, com o objetivo de simplificar o desenvolvimento de aplicações. Nesse sentido, foi desenvolvida para a linguagem de programação em *Java*, pelo que se baseia nos conceitos de inversão de controlo (*IOC*) e injecção de dependências. Para além disso, esta *framework* apresenta diversos módulos que podem ser utilizados de acordo com as necessidades do projeto, como por exemplo *Spring Data* (trata da persistência) e *Spring Security* (trata da segurança da aplicação). Assim, a Spring foi criada com intuito de combater um estilo de programação pesado e extremamente dependente de várias configurações.

Características da *Spring*

1. Uma das principais características desta *framework* é o facto de esta não precisar de um servidor de aplicação para funcionar. Apenas utiliza *JVM*, pelo que assim, possibilita acesso a recursos que anteriormente só estariam disponíveis para soluções corporativas;
2. Permite utilizar apenas aquilo que é necessário para o projeto. Esta *framework* permite abandonar o conceito *EJBs* (que levavam a implementar comportamentos que não eram necessários), pelo que permite

construir uma arquitectura mais leve, fácil de compreender, manter e consequentemente de evoluir;

3. Outra diferença é o facto desta *framework* ser baseada na inversão de controlo e injeção de dependências, pelo que fornece para isso um *container*, que representa o núcleo da *framework* e que é responsável por criar e gerir os componentes da aplicação, conhecidos por *beans*.

Vantagens

- Fácil de testar;
- Liberdade para trabalhar na *View*;
- Integração simples com outras *frameworks*;
- Suporte a várias *Views Freemaker*, *JSP* e *Velocity*;
- Permite simplificar código.

Desvantagens

- Apresentação dos erros (o facto de aparecer em código, por exemplo 400, 404 e 500);
- A cadeia de objetos pode-se tornar extensa.

Spring Design Patterns

A *framework Spring* faz referência a *Design Patterns* pelo que, de seguida, serão apresentados alguns exemplos:

- ***Proxy Design Pattern*:** Uma classe é usada para representar a funcionalidade de outra classe. Trata-se de um exemplo de padrão estrutural. Um objeto é criado com um objeto original para relacionar a sua funcionalidade com o mundo externo;
- ***Singleton Design Pattern*:** Garante que existirá apenas uma instância única de um objeto na memória que pode fornecer serviços.

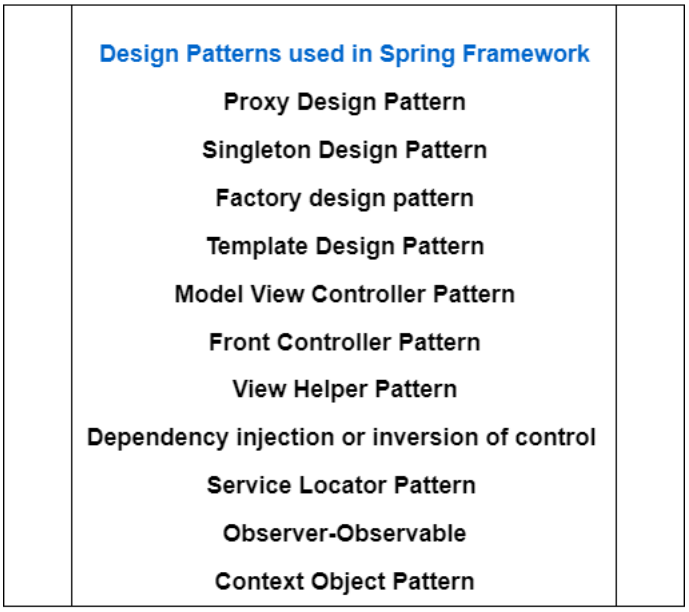

Figura 4: *Design Patterns* utilizados na *Spring framework*

3.2.2 *CSLA*

O *CSLA .NET* é uma estrutura de desenvolvimento de *software*, criada por Rockford Lhotk, que auxilia na criação de uma camada de negócios orientada a objetos, reutilizável e sustentável. Os objetos baseados no *CSLA* possuem muitos recursos avançados que simplificam a criação de interfaces Windows, web, entre outros. Para além disso, o *CSLA .NET* permite grande flexibilidade na persistência de objetos, de modo que estes podem usar virtualmente qualquer fonte de dados disponível. Nesse sentido, apresentam-se de seguida algumas das suas vantagens e desvantagens.

Vantagens

- Permite flexibilidade no que diz respeito a gerir várias fontes de dados e protocolo de rede;
- Possui classes bastante úteis;
- Permite reduzir a sobrecarga da aplicação.

Desvantagens

- Requer tempo para aprender;
- Exige uma ampla familiaridade com a *framework*;
- Requer experiência em programação orientada por objetos.

Em suma, a *CSLA* é bastante útil pois permite melhorar o desempenho, a escalabilidade, a segurança e a tolerância a falhas das aplicações, sem alterações no código na interface do utilizador ou nos objetos.

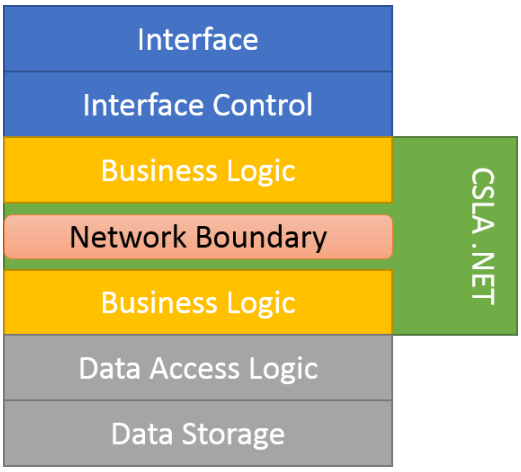

Figura 5: Estrutura da *CSLA*

3.3 *Frameworks* para Camada de Apresentação

A utilização de *frameworks* para a camada em questão torna-se relevante devido ao facto de estas facilitarem o *deploy* e o desenvolvimento de aplicações web, simplificando a codificação de certas funcionalidades, como, por exemplo, a gestão de sessões.

3.3.1 *Spring MVC*

A *framework Spring MVC*, incluída na *framework Spring*, mencionada na camada de negócio, utiliza uma arquitetura *Model-View-Controller* (*MVC*), observando-se assim uma separação entre os diferentes aspetos da aplicação,

nomeadamente, do *input*, da lógica de negócio e da apresentação. Esta prática faz com que seja possível desenvolver aplicações web flexíveis e *loosely coupled*.

Esta *framework* utiliza um *Dispatcher Servlet*, que se encontra responsável por receber os pedidos *HTTP* do cliente e devolver-lhe a página associada. Sendo assim, pode-se afirmar que esta *framework* funciona como *server-side*. Este funcionamento encontra-se exemplificado na figura apresentada de seguida.

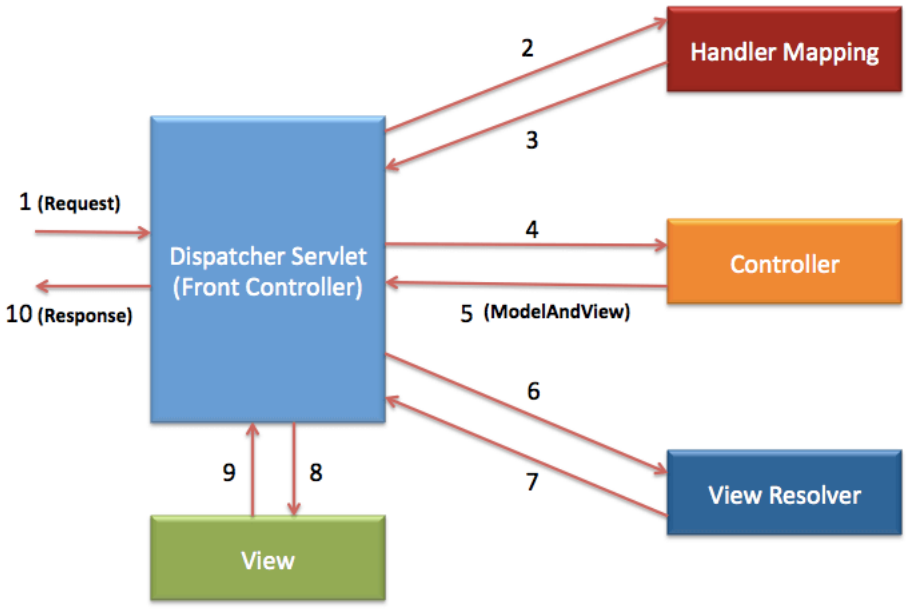

Figura 6: Arquitetura da *Spring MVC*

Tal como se pode verificar pela figura apresentada a cima, o procedimento que a *Spring MVC* executa é composto, mais concretamente, pelos seguintes passos:

1. Após receber o pedido *HTTP*, o *Dispatcher Servlet* consulta o *Handler Mapping* para poder chamar o *Controller* associado ao pedido;

2. Por sua vez, o *Controller* lê o pedido e chama os métodos apropriados consoante o método *GET* ou *POST* utilizados. O modelo de dados

retornado ao *Dispatcher Servlet* será baseado no que está definido na lógica de negócio, retornando-se também o nome da *View*;

3. O *Dispatcher Servlet* com a ajuda do *View Resolver* identifica a *View* requisitada pelo pedido;

4. Quando a *View Resolver* finaliza, o *Dispatcher Servlet* passa o modelo à *View*, que posteriormente é renderizada no *browser*.

Vantagens

- Integra-se facilmente com a *framework Spring*;
- Documentação e comunidade acessível;
- Permite a escrita de código mais limpo e percetível;
- Módulos *loosely coupled*;
- Uso flexível de *DI* (*Dependency Injection*);
- Modularidade;
- Testes aos dados são simplificados com a utilização de *POJOs* (*Plain Old Java Objects*);
- Enumeras funcionalidades para qualquer projeto que se pretenda desenvolver, como, por exemplo, *SOAP services* e *REST APIs*.

Desvantagens

- Tempo demorado de aprendizagem por isso recomenda-se que já se tenha alguns conhecimentos sobre a utilização da *framework*;
- Arquitetura *MVC* pode ser difícil de implementar para pessoas inexperientes na interação com a mesma.

3.3.2 *JSF*

A *JavaServer Faces* (*JSF*) é uma *framework* utilizada para desenvolver aplicações web programadas em *Java*, sendo, mais concretamente, uma *framework* para a camada de apresentação. Desenvolvida pela *Java Community Proccess*, esta implementa o modelo de arquitetura *MVC* (*Model-View-Controller*) e é uma *framework server-side* orientada a eventos. Para além disso, nesta foca-se o desenvolvimento de *user interfaces* (*UI*) baseadas em componentes pré-definidos e reutilizáveis.

O *JSF* utiliza *Facelets*, que é um sistema de *template* web, como sistema de *template* padrão que utiliza arquivos *XML*, aos quais se dá o nome de *Facelets views* ou modelos de visão. Para a criação de modelos de visão são necessários seis passos, que correspondem ao ciclo de vida do *JavaServer Faces*, tal como se apresenta na imagem que se segue.

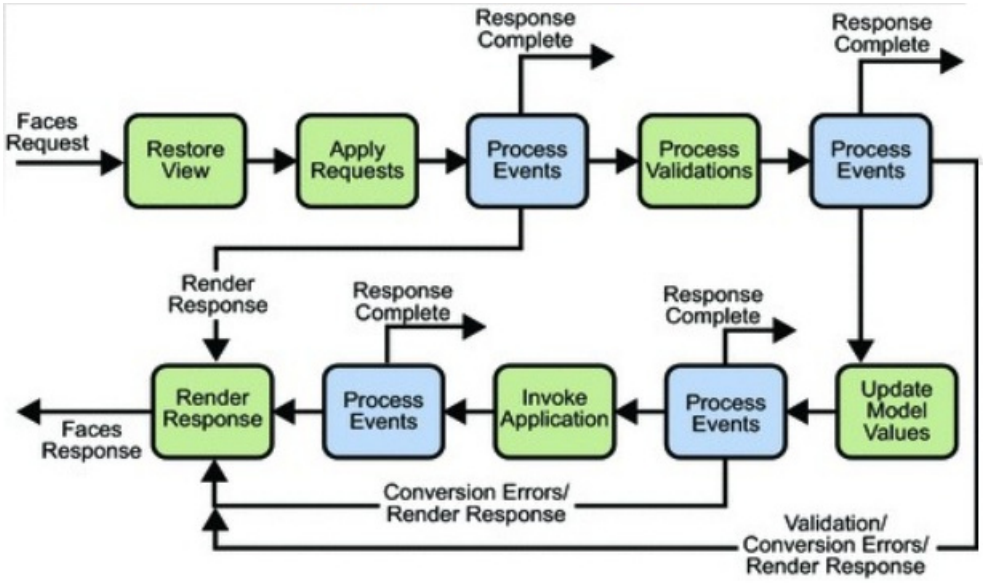

Figura 7: Ciclo de vida da *JSF*

Por último, uma das principais características desta *framework* é o encapsulamento das tecnologias *client-side*, tal como *HTML* e *JavaScript*, o que permite que o utilizador possa desenvolver a aplicação web sem ter que interagir com estas. Todas estas características resultam num conjunto de vantagens e desvantagens, sendo que se apresentam algumas em seguida.

Vantagens

- Permite a criação de páginas web dinâmicas;
- Simplifica a criação de *views*;
- Facilita a transferência de dados através da camada de modelo;
- Permite a comunicação entre as camadas de controlo e de visão da aplicação web, através de *managed beans*;
- Faz a separação de funções na construção da aplicação, permitindo que as camadas de negócio e apresentação possam ser desenvolvidas separadamente;
- Apresenta um conjunto de componentes pré-definidos;
- Reutiliza componentes da página;
- Não exige conhecimento aprofundado das tecnologias *client-side*.

Desvantagens

- Apresenta uma má curva de aprendizagem, sendo necessário despender bastante tempo para dominar esta *framework*;
- A aprendizagem demorada leva a que não seja ideal para projetos pequenos, curtos prazos de entrega e programadores inexperientes;
- Rigidez na abordagem de desenvolvimento, devido ao modelo *MVC*;
- O encapsulamento das tecnologias e a abstração fornecida pelos componentes podem dificultar o *debug* da aplicação.

É também interessante referir que esta *framework* tira partido de vários *design patterns* na sua arquitetura, tal como o *Singleton Pattern*, *MVC Pattern*, *Factory Method Pattern*, *State Pattern*, *Composite Pattern*, *Decorator Pattern*, *Template Method Pattern*, entre outros.

3.3.3 *GWT*

A *Google Web Toolkit* (*GWT*) é uma *framework open-source* para o desenvolvimento da camada de apresentação de aplicações web programadas em *Java*. Esta é uma *framework* de solução mista, que permite fazer o balanceamento da carga entre o cliente e o servidor, pois baseia-se nos modelos de *stateless server* e *stateful client*, em que a informação do cliente é guardada no seu lado, ao invés de o servidor ter que armazenar as informações de todos os clientes.

A principal caraterística desta *framework* é o compilador que esta contém, que compila o código *Java* em *JavaScript*, *HTML* ou *CSS*, não sendo necessário que o utilizador saiba manipular tecnologias *client-side*. Deste modo, toda a aplicação web pode ser programada em *Java*, a única linguagem que o utilizador necessita de dominar. Para além disso, este compilador otimiza o código, remove código desnecessário e ofusca-o, tornando difícil a sua manipulação por parte do utilizador.

Numa outra vertente, também importante, a *GWT* facilita o *debug* das aplicações, permitindo visualizar a página que está a ser analisada através de um *browser*, sendo apenas necessário instalar um *plugin* próprio, e permite que o *debug* e teste da aplicação possa ser feito em qualquer IDE à escolha do utilizador, tal como uma aplicação local.

Vantagens

- Adequada para aplicações web em larga escala;
- Lida com a compatibilidade dos *browsers*;
- Os componentes *UI* são dinâmicos e reutilizáveis;
- Existe bastante documentação sobre a mesma;
- A aprendizagem é bastante simples;
- O utilizador não necessita de ser experiente nas ferramentas *client-side*, necessitando apenas de ter conhecimentos em *Java*;
- Torna o *debug* da aplicação prático;
- É compatível com bastantes IDE's, tal como o IntelliJ.

Desvantagens

- Dificuldade em manipular e costumizar o *JavaScript*, *HTML* e *CSS*;
- Pode chegar a tempos de compilação elevados;
- Não disponibiliza uma lista de componentes pré-definidos.

3.3.4 *Vaadin*

Vaadin é uma *framework open-source* utilizada para desenvolver a camada de apresentação de aplicações web em *Java*. Esta *framework* pode ser utilizada para uma arquitetura *server-side* ou *client-side*, sendo uma *framework* de solução mista. No entanto, o modelo *server-side* é o mais adotado e o mais eficaz, permitindo que seja utilizado um *engine* de *AJAX* no lado do cliente para renderizar a interface no mesmo.

Sendo uma *framework* que se foca no desenvolvimento de *user interfaces* (*UI*), esta apresenta uma grande variedade de componentes pré-definidos, otimizados para otimizar o desenvolvimento, a experiência do utilizador e a escalabilidade.

Uma das caraterísticas mais importantes da *Vaadin* é o utilizador programar as interfaces sem ter que se preocupar com a parte web, pois todas as comunicações *AJAX* entre o cliente e o servidor são encapsuladas. Para além disso, apresenta um compilador que trata da transformação de *Java* em *JavaScript* e que gera *HTML*, não sendo necessário o utilizador estar familiarizado com mais alguma linguagem para além do *Java*.

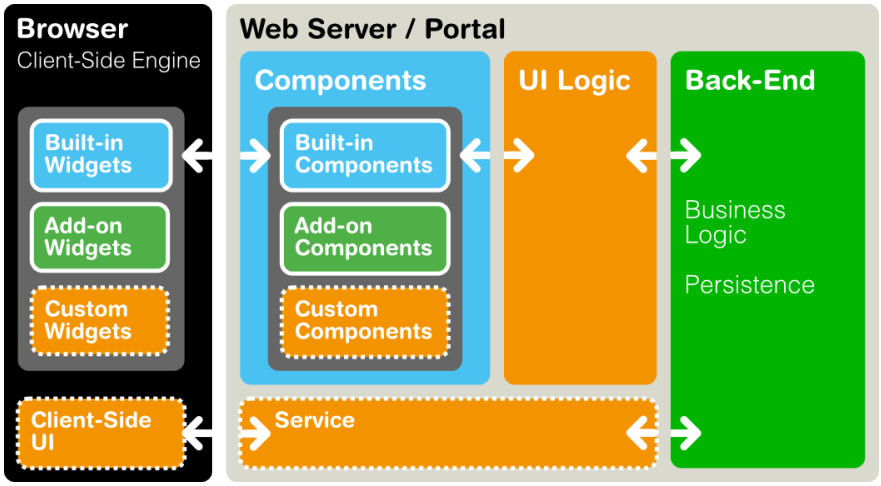

Figura 8: Arquitetura de uma aplicação em *Vaadin*

Vantagens

- Possibilidade de programar apenas o lado do servidor, sendo a *framework* que trata do lado do cliente;
- Comunicação entre cliente e servidor automática;
- Apresenta um conjunto de componentes pré-definidos e temas com atenção ao design;
- Permite desenvolver *Rich Internet Applications*, isto é, aplicações bastante interativas e responsivas, quase como aplicações locais;
- Integra-se facilmente com a *framework Spring*;
- Não exige conhecimento aprofundado de tecnologias *client-side*, sendo apenas necessário dominar o *Java*.

Desvantagens

- Torna complicado customizar a interface, devido ao *HTML* e *CSS* serem gerados automaticamente;
- Facilmente o código se torna bastante extenso e complexo.

3.3.5 *ASP.NET*

A *ASP.NET* é uma *framework server-side open-source*, que pertence à *Microsoft* e estende a *framework .NET*, através da adição de ferramentas próprias para construção de aplicações web dinâmicas. Desta forma, esta *framework* apresenta-se como sucessor da *framework ASP*.

Vantagens

- Simplificação no desenvolvimento de aplicações através da utilização de *Language Integrated Query* (*LINQ*) e programação assincrona;
- Aplicações mais rápidas e escaláveis;
- Utilização de serviços de segurança que previnem problemas como alteração de apostadores ou manipulação indevida do código;
- *Open-source*;
- Comunidade ativa sendo que se encontra facilmente ajuda *online*;
- *Cross Platform*, isto é, nas aplicações desenvolvidas com *.NET* é possível utilizar *C#*, *F#*, ou *Visual Basic*, sendo que o código pode ser reutilizado mesmo tendo sido escrito com linguagens de programação diferentes;

Desvantagens

- Opera apenas em *Windows*.

Além da *ASP.NET*, é importante realçar a existência da *ASP.NET MVC* e da *ASP.NET Core*. A *ASP.NET MVC*, tal como o nome indica, utiliza a arquitetura *MVC* para criação de aplicações web, em vez de utilizar *web forms* e *postbacks*, como acontece na *ASP.NET*. A *ASP.NET Core* é o sucessor da *ASP.NET*, sendo que este permite unificar a *ASP.NET* e a *ASP.NET MVC* num só modelo de programação. Além disso, esta permite que as aplicações possam ser executas em diferentes plataformas, nomeadamente, *Windows*, *Mac* e *Linux*, ao contrário da *ASP.NET* que só permite *Windows*.

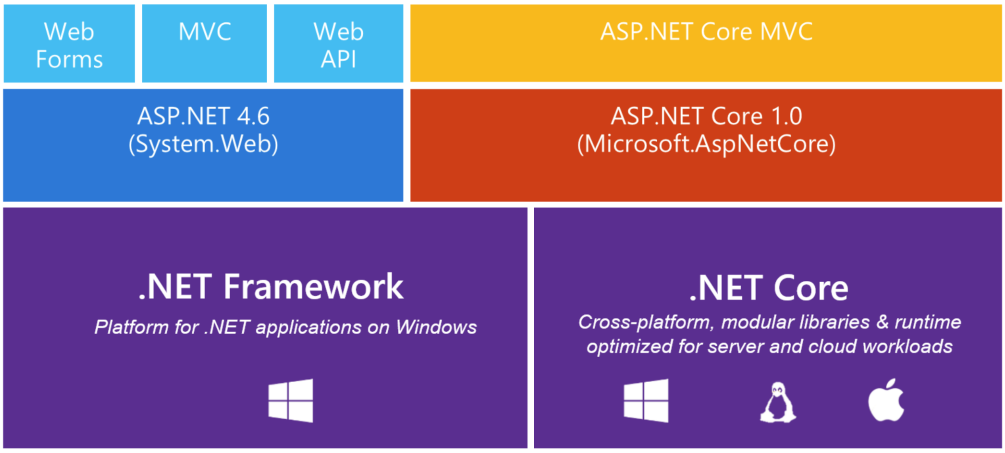

Figura 9: Comparação entre *ASP.NET* e *ASP.NET Core*

4 Arquitetura Final

Como último capítulo irá-se apresentar uma proposta para uma arquitetura que poderá vir a ser utilizada no trabalho prático desta unidade curricular. Nesse sentido, como linguagem preferida para esta, decidimos utilizar *Java* pois tem diversas vantagens em relação a outras (como a *.Net*), nomeadamente:

- *Java* é a linguagem mais utilizada na indústria de *Software*;
- Integração com inúmeras plataformas;
- Portabilidade em relação ao Sistema Operativo;
- Migração para outra plataforma muito simplificada;
- Escolha entre várias *IDEs*.

4.1 Camada de Dados

Como *framework* para a camada de dados o grupo escolheu *Hibernate*. De facto, a curva de aprendizagem pode ser um fator negativo, mas a grande quantidade de documentação online equilibra a decisão. Um outro fator

muito importante na decisão foi o facto desta ferramenta usar uma linguagem própria o que torna o código independente do motor de Base de Dados, podendo alterar o motor sem qualquer alteração no código. Por fim, todas as outras características do *Hibernate*, tornam-no numa das melhores *frameworks* para desenvolvimento em *Java*.

4.2 Camada de Negócio

Relativamente à camada de negócio optamos por escolher a *framework Spring*. Um dos aspetos que nos levou a escolher esta foi a sua fácil conexão com a *framework Hibernate*, que foi por nós escolhida para a camada de dados. Além disso, as pesquisas efectuadas na primeira fase, levaram a perceber que se trata de uma *framework* bastante utilizada e com uma elevada quantidade de vantagens. Para além deste facto, acreditamos que as desvantagens consideradas não têm demasiado impacto, pelo que serão facilmente contornadas.

4.3 Camada de Apresentação

Tendo em conta que se pretende desenvolver uma arquitetura em *Java*, ficamos perante quatro opções: *Spring MVC*, *JSF*, *GWT* e *Vaadin*. Todas estas possuem vantagens atraentes, no entanto, devido à desvantagem apontada no *Spring MVC* e *JSF* sobre a dificuldade de aprendizagem, decidiu-se descartar essas duas opções. Comparando então *GWT* e *Vaadin*, estas são bastante semelhantes. No entanto, optou-se por escolher a *GWT*, por este permitir um bom balanceamento entre o cliente e o servidor, não pesando tanto o segundo, faz uma melhor otimização do código e, ao contrário de todas as outras frameworks, facilita o *debug* das aplicações web em IDE's, tal como se fosse uma aplicação de *desktop*. Para além disto, apesar de ser bastante poderosa, a *GWT* apresenta uma excelente curva de aprendizagem, sendo bastante simples de dominar. É também importante referir que esta *framework* escolhida é compatível com a *Spring*, a *framework* selecionada para a camada de negócio.

Para além destes pontos, também se teve em consideração tentar optar por uma *framework* de solução mista, pois consideramos ser a mais vantajosa, não sobrecarregando tanto o servidor e permitindo obter interfaces mais interativas.

4.4 Desenho da Arquitetura

Tendo em conta as *frameworks* escolhidas para as várias camadas, apresenta-se de seguida uma esquematização da arquitetura proposta.

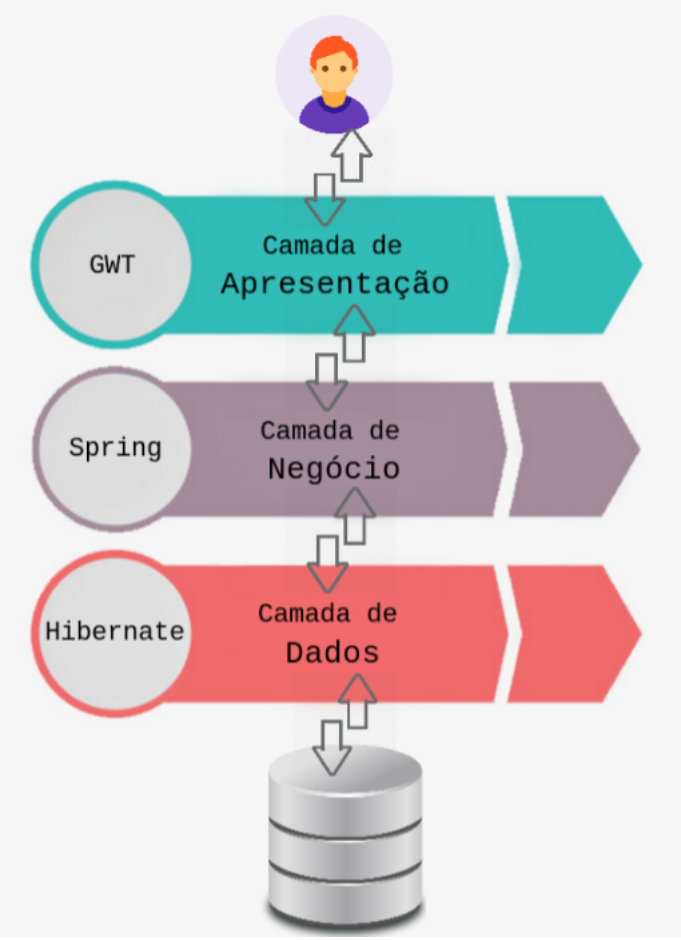

Figura 10: Arquitetura proposta

5 Conclusão

Na resolução deste trabalho foi possível entrar em contacto com o conceito de *framework* e quais as vantagens que estas trazem para as diferentes camadas de uma aplicação. Com este estudo, deparamo-nos com a existência de várias *frameworks* disponíveis no mercado, prontas a utilizar, sendo algumas mais populares que outras, e concluímos que a quantidade de *frameworks* para cada camada também difere, sendo a de apresentação a que continha maior diversidade.

Além disso, no processo de tentar associar *design patterns* às *frameworks* e definir estas como *client-side*, *server-side* ou *hybrid*, houve alguma dificuldade devido ao facto da informação pelos vários *websites* discrepar, sendo que o grupo não pode obter conclusões 100% confiáveis quanto a este tópico.

Apesar da arquitetura definida, compreendemos que esta poderá sofrer alterações quando realmente utilizar-mos as *frameworks* selecionadas, pois estas foram escolhidas baseadas na opinião de outros utilizadores. Em jeito de conclusão, apresentamos de seguida o conjunto de *frameworks* utilizadas, independentes da linguagem, para desta forma, dar a conhecer o panorama geral e atual destas.

Rankings

Framework	Github Score	Stack Overflow Score	Overall Score
AngularJS	93	97	95
React	99	92	95
Ruby on Rails	90	99	94
ASP.NET MVC		94	94
Angular	91	93	92
Django	90	95	92
Laravel	91	91	91
Vue.js	100	80	90
ASP.NET	78	100	89
Spring	86	93	89

Figura 11: Ranking das *Frameworks* mais utilizadas com base no score do *GitHub* e *StackOverFlow*

Referências

[1] https://javapipe.com/blog/best-java-web-frameworks/

[2] https://spin.atomicobject.com/2015/04/06/
web-app-client-side-server-side/

[3] https://www.sqa.org.uk/e-learning/ClientSide01CD/page_18.htm

[4] https://www.oreilly.com/library/view/java-server-pages/
156592746X/ch12s04.html

[5] http://hibernate.org/orm/

[6] https://www.java4s.com/hibernate/main-advantage-and-disadvantages-of-hibernate

[7] https://www.tutorialspoint.com/mybatis/mybatis_useful_
resources.html

[8] https://www.tutorialspoint.com/mybatis/mybatis_hibernate.
html

[9] https://www.itprotoday.com/development-techniques-and-management/
5-reasons-why-entity-framework-can-be-your-best-friend

[10] https://www.devmedia.com.br/exemplo/
como-comecar-com-spring/73

[11] https://spring.io/

[12] http://www.informit.com/articles/article.aspx?p=770361&
seqNum=4

[13] https://en.wikipedia.org/wiki/Web_framework

[14] https://javapipe.com/blog/best-java-web-frameworks/

[15] https://docs.spring.io/spring/docs/current/
spring-framework-reference/web.html

[16] https://www.tutorialspoint.com/spring/spring_web_mvc_
framework.htm

[17] https://www.javaworld.com/article/3322533/enterprise-java/what-is-jsf-introducing-javaserver-faces.html

[18] https://pt.wikipedia.org/wiki/JavaServer_Faces

[19] https://www.slideshare.net/JorgeWilliamRodrigues/introduo-a-jsf

[20] https://www.ibm.com/developerworks/library/wa-dsgnpatjsf/index.html

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

[22] https://en.wikipedia.org/wiki/Google_Web_Toolkit

[23] http://thoughtfulsoftware.blogspot.com/2013/03/pros-and-cons-of-google-web-toolkit.html

[24] http://thoughtfulsoftware.blogspot.com/2013/03/pros-and-cons-of-google-web-toolkit_23.html

[25] http://wc.demo.vaadin.com/mcm/out/framework/introduction/intro-overview.html

[26] https://vaadin.com

[27] https://developer.ibm.com/dwblog/2017/what-is-vaadin-java-web-applications/

[28] https://en.wikipedia.org/wiki/ASP.NET

[29] https://dotnet.microsoft.com/apps/aspnet

[30] https://dotnet.microsoft.com/platform/why-choose-dotnet

[31] https://en.wikipedia.org/wiki/ASP.NET_Core

[32] https://stackify.com/java-vs-net-comparison/

[33] https://hotframeworks.com/