Universidade do Minho

Mestrado em Engenharia Informática

Arquiteturas Aplicacionais

Trabalho I

Frameworks das Camadas Aplicacionais

Ana Murta (PG50184)

Beatriz Oliveira (PG50942)

Gonçalo Soares (PG50393)

Joana Alves (PG50457)

Vicente Moreira (PG50799)

Fevereiro 2023

Conteúdo

1 Introdução

Neste relatório, apresentamos o resultado da nossa pesquisa, enquanto grupo, relativamente às *frameworks* existentes de separação de camadas.

As frameworks são ferramentas versáteis, robustas e eficientes que auxiliam o desenvolvimento de aplicações, uma vez que permitem aos programadores focarem-se nas funcionalidades *high level*, sendo que as funcionalidades de *low-level* são tratadas pelas mesmas.

Assim, para cada uma destas iremos explicar o seu funcionamento, as vantagens relativas à sua utilização e, ainda, as distinções existentes entre as diversas frameworks que apresentamos.

Para além disso, realizamos, também, uma discussão relativamente às frameworks que são exclusivamente *server side* e as que são híbridas e apresentamos uma proposta de arquitetura genérica que utiliza algumas das *frameworks* apresentadas.

2 *Frameworks*

De seguida, para cada camada aplicacional (**Dados**, **Lógica de Negocio** e **Apresentação**), apresentamos três *frameworks* diferentes, explicitando, tal como referido anteriormente, os seus pontos fortes e/ou o que as distinguem no mercado.

2.1 Camada de Dados

2.1.1 Hibernate

O Hibernate é uma framework, escrita em *Java*, que simplifica o desenvolvimento de aplicações desta linguagem para interagir com bases de dados. Trata-se de uma ferramenta *ORM* (*Object Relational Mapping*) *open source*, que implementa as especificações da *JPA* (*Java Persistence API*) para a persistência dos dados. Algumas das suas vantagens são:

- **Performance rápida -** Isto deve-se à utilização interna da cache na estrutura desta framework. A cache mantém uma cópia dos objetos no servidor da aplicação, permitindo que sejam evitadas consultas repetitivas e dispendiosas da base de dados.
- **Query independente da base de dados -** HQL, *Hibernate Query Language*, é uma versão orientada a objetos do SQL, que gera queries independentes da base de dados. Não sendo necessária a escrita de queries específicas para a base de dados, já que, antes desta framework, caso houvesse uma mudança de base de dados, num dado projeto, era necessário a alteração das queries, o que conduzia a um problema de manutenção.
- **Criação automática de tabelas -** Esta ferramenta fornece a facilidade de criar as tabelas das bases de dados automaticamente.
- **Simplifica operações de *join* complexas**
- **Fornece estatísticas de consulta e de estado da base de dados -** Uma vez que suporta *Query Cache*, o Hibernate fornece estatísticas relativas à consulta e ao estado da base de dados.

2.1.2 MyBatis

MyBatis é uma framework de persistência que automatiza o mapeamento entre as bases de dados SQL e os objetos em *Java*, *.NET* e *Ruby on Rails*. Este mapeamento é desacoplado da lógica da aplicação ao compactar as instruções SQL em ficheiros de configuração XML.

Comparativamente a outras frameworks persistentes, a principal distinção do MyBatis é que este enfatiza a utilização de SQL, enquanto que as restantes utilizam, normalmente, uma linguagem de consulta personalizada. Assim sendo, as vantagens de utilizar esta framework são:

- **Simplicidade -** Para além do MyBatis fornecer uma API simples para interagir com a base de dados, esta é amplamente considerada como uma das frameworks de persistência mais simples relativamente às que temos, atualmente, disponíveis.
- **Portabilidade -** Esta pode ser implementada para praticamente qualquer linguagem ou plataforma.

- **Suporta procedimentos armazenados** - O MyBatis encapsula SQL na forma de procedimentos armazenados para que exista a possibilidade da lógica de negócios não estar contida na base de dados, permitindo aumentar a portabilidade da aplicação e facilitar a sua implementação e os testes da mesma. Além disso, suporta SQL personalizado e mapeamentos avançados.

- **Suporta *inline* SQL -** O utilizador tem o acesso total a todos os recursos do SQL, não existindo necessidade de um pré-compilador.

- **Suporta SQL dinâmico -** São fornecidos recursos para queries SQL de construção dinâmica com base em parâmetros.

- **Suporta ORM -** É oferecido suporte a muitos dos recursos de uma ferramenta ORM, tal como o carregamento lento, o armazenamento em cache, a geração de código em tempo de execução e a herança.

2.1.3 Prisma

Prisma é uma framework *ORM* (Object-Relational Mapping) open-source para *Node.js* e *TypeScript*, que permite uma interação facilitada, intuitiva e *type-safe* de vários tipos de bases de dados, como *MySQL*, *PostgreSQL*, *SQLite* e *MongoDB*.

Esta framework consiste em 3 partes: ***Prisma Client*** responsável pela construção automática e type-safe de queries. ***Prisma Migration***, uma ferramenta de migração de bases de dados e, finalmente, o ***Prisma Studio***, um editor visual para a manipulação de dados.

Todos estes componentes para funcionarem necessitam de um *Prisma Schema*, um ficheiro de configuração principal escrito em *DSL* que especifica os detalhes acerca da fonte dos dados acedidos, o seu modelo de dados e os clientes que irão aceder à base de dados.

Algumas vantagens na utilização desta framework são:

- **Deteção de Erros -** Devido ao seu ênfase de type-safety e através da declaração do modelo de dados no Prisma Schema, é possível encontrar erros de tipagem de dados na fase de compilação da aplicação, evitando assim erros de execução.

- **Fácil gestão e manutenção -** Através das ferramentas fornecidas de seeding, migração e limpeza, é facilitado o processo de manutenção da base de dados.

- **Queries potentes -** Graças ao conhecimento do modelo de dados, é possível formular e executar queries complexas com várias condições, sortings e joins.

- **Documentação e comunidade -** Visto que se trata de uma framework open-source, é possível encontrar vários contribuidores e suporte através de fóruns e de toda a documentação disponibilizada.

2.2 Camada de Lógica de Negócio

2.2.1 Spring

Spring é uma *open-source framework* que fornece suporte de infraestrutura no desenvolvimento de aplicações Java. Esta foi lançada em junho de 2003 por Rod Johnson sob a licença Apache 2.0 e organizada pela *SourceForge*. A *Spring* remove o trabalho aborrecido de configuração, permitindo aos programadores concentrarem-se na parte lógica da aplicação.

Esta framework apresenta ser umas das mais utilizadas pelos programadores devido às seguintes vantagens:

- **Modularidade e Flexibilidade** - A *Spring* é uma *framework* modular, ou seja, é dividida em vários módulos, podendo estes ser usados individualmente ou em conjunto. Isto permite escolher apenas os módulos relevantes para desenvolvimento da aplicação, tornando o código mais eficiente.
- ***Light Weight*** - Isto deve-se à sua implementação *POJO* (*Plain Old Java Object*), que permite o desenvolvimento de aplicações escaláveis e a simplificação do mesmo, aumentando também o desempenho.
- **Injeção de dependências** - A *Spring* possibilita que as dependências de uma classe sejam fornecidas por outros objetos, em vez de serem criadas pela própria classe, o que simplifica a configuração e gerência dos objetos.
- **Desenvolvimento de testes mais fáceis e rápidos** - Proporciona uma identificação e correção mais rápida de problemas no código, visto que utiliza injeção de dependências e simulação de objetos em testes unitários.

2.2.2 Django

Django é uma open-source *framework* baseada em *Python* com ênfase na reutilização e modularidade de componentes, com base na arquitetura model–template–views (MTV). Esta foi criada em 2003 por Adrian Holovaty e Simon Willison com o objetivo de auxiliar o desenvolvimento de aplicações em Python e, depois do seu lançamento em 2005, foi criada a Django Software Foundation que mantém a framework atualizada.

Esta framework revela-se popular devido as várias componentes incluídas e as suas vantagens tais como:

- ***Lightweight*** **e Desenvolvimento Rápido -** Django possui várias funcionalidades para permitir a criação rápida de web servers para desenvolvimento e teste, assim como API's incluídas com funções comuns e úteis de caching, routing e autenticação.
- **Python ORM -** Permite a fácil leitura e manipulação de bases de dados através de objetos Python, existindo suporte para as várias relações presentes nas tabelas e até a sua modificação.
- **Escabilidade -** A modularidade do Django facilita a escalabilidade das aplicações desenvolvidas, sendo possível escalar estas verticalmente ou horizontalmente.

- **Sistema de serialização -** É possível converter e manipular automaticamente estruturas de dados complexas para formatos de fácil transmissão e/ou armazenamento. Assim sendo, é possível converter Python Objects ou dados de uma base de dados para formatos como JSON, XML ou YAML e vice-versa.

- **Segurança -** Django possui componentes base capazes de prevenir e mitigar ataques comuns na Web como SQL Injection, cross-site scripting (XSS) e cross-site request forgery (CSRF).

- **Comunidade -** Devido à sua natureza extensível e suporte para a *pluggability* de componentes, existe uma grande comunidade de contribuidores ativos que suportam e desenvolvem novas features para a framework. Graças a esta comunidade extensa, é realizada uma conferência semi-anual chamada **DjancoCon**.

2.2.3 Laravel

Laravel é uma *open-source PHP web framework* para o desenvolvimento de aplicações que seguem a arquitetura MVC (*model-view-controlle*r). A sua primeira versão foi lançada em junho de 2011 por Taylor Otwell. Contudo, esta apenas incluía suporte a *models* e a *views*. Somente na segunda versão foi adicionado o suporte aos *controlleres*. Nos anos seguintes, foram lançadas novas versões com novos recursos e atualizações.

Algumas vantagens desta *framework* são:

- **Autenticação** - *Laravel* permite uma implementação bastante simples da autenticação uma vez que oferece uma fácil configuração. Esta *framework* fornece uma organização lógica de autorização simples e um fácil controlo dos recursos de acesso. Esta vantagem permite controlar o acesso a recursos protegidos, processar mais facilmente os pedidos de acesso e rejeitar os pedidos não autorizados.

- **API simples** - Esta *framework* oferece uma API simples que funciona de forma fluída com a biblioteca *SwiftMailer*. Esta biblioteca possibilita a simplificação da configuração do mail API para contas individuais. Com a biblioteca integrada, *Laravel* suporta o envio de emails e notificações a partir de múltiplos canais de entrega. Além disto, fornece controladores para vários serviços de email locais e baseados em nuvem, tais como *Mandrill*, *SMTP*, *Mailgun*, *Amazon SES*, *SparkPost* e função de PHP mail.

- **Cache *backends*** - Oferece suporte à cache do lado do servidor com drivers incorporados, que armazenam, num sistema de ficheiros, objetos em cache. Isto proporciona um melhor desempenho do *backend*, uma maior rapidez de carregamento e utilização da aplicação, e uma melhor gestão de memória.

- **Gestão de erros e exceções** - *Laravel* contém um módulo de tratamento de erros e exceções. Para além disto, ainda inclui a biblioteca de registo *Monolog* que facilita a gestão de erros através do fornecimento de suporte a vários gestores *log* de erros e exceções. Isto permite uma interface informativa e de fácil utilização, que se traduz num aumento da satisfação do cliente. Por fim, no caso das exceções, também melhora a entrega de informação ao utilizador em tempo real.

- **Testes** - Esta *framework* tem métodos de teste e de suporte de teste *PHPUnit* usando um ficheiro *phpunit.xml*, o que permite testar facilmente o comportamento fundamental do utilizador. Proporcionando testes mais rápidos devido à automatização e ao cálculo do desempenho da aplicação através do processamento de múltiplos cenários.
- **Flexibilidade** - *Laravel* contém extensas bibliotecas pré-instaladas, possibilitando criar e manter uma variedade de funcionalidades para a aplicação. A flexibilidade desta *framework* permite-lhe escalar aplicações de qualquer complexidade e dimensão.

2.3 Camada de Apresentação

2.3.1 *React*

A biblioteca *React JS* foi lançada pelo *Facebook* em 2013, com o objetivo de otimizar a atualização e a sincronização de atividades simultâneas no *feed* de notícias, como o *chat*, o *status*, entre outros, sendo uma das bibliotecas mais populares atualmente. Como vantagens desta biblioteca podemos realçar:

- ***Virtual DOM* -** Ao utilizar *Virtual DOM*, o *React* utiliza uma estratégia mais declarativa, isto é, contrariamente ao DOM apenas executa *re-render* ao elemento em causa, não fazendo *re-render* aos filhos do mesmo, demonstrando vários ganhos significativos, por exemplo, em termos de desempenho.
- ***Single Page Application* -** As páginas criadas com esta bilbioteca são SPA's, ou seja, grande parte da computação por parte do UI é feita no lado do cliente. Isto pode ser positivo no sentido em que a aplicação aparenta ser nativa e oferece uma experiência mais suave ao utilizador.
- **Funcionalidades -** Uma vez que o principal objetivo do *React* é facilitar o desenvolvimento de interfaces, esta biblioteca possui diversas funcionalidades que aumentam a produtividade e auxiliam no processo de desenvolvimento.
- **Comunidade -** Como é uma das bibliotecas mais utilizadas na área, possui uma comunidade e um suporte enorme, permitindo aos novos utilizadores uma rápida aprendizagem e resolução de problemas.

Relativamente a desvantagens da biblioteca *React* podemos denotar as seguintes:

- **SPA -** As *Single Page Applications* acarretam alguns problemas de compatibilidade entre *browsers* e de SEO (*Search Engine Optimization*).
- **SEO -** É considerado um desafio uma vez que *web crawlers* não conseguem ler os metadados das diferentes páginas que encontram-se embutidas numa só.
- ***Browsers* -** Em termos de compatibilidade entre *browsers*, como o UI é gerado do lado do cliente, alguns funcionalidades de *Javascript* ou *HTML* podem não estar disponíveis no *browser* utilizado.

2.3.2 *Vue*

Vue JS é uma *framework Javascript open-source*, lançada em Fevereiro de 2014 por Evan You, programador na *Google Creative Labs*, com o objetivo de agilizar o processo de desenvolvimento de interfaces gráficas. Algumas das vantagens ou pontos distintivos desta *framework* são:

- **Componentes -** Permite construir interfaces com um conjunto de componentes reutilizáveis, tornando o código mais organizado e facilitando a sua manutenção.
- **Reatividade -** Utiliza o sistema de reatividade do *JavaScript* para atualizar automaticamente a interface quando os dados mudam.

- ***Virtual DOM* -** Utiliza Virtual DOM, tal como a biblioteca de *React* e como foi referido anteriormente, apenas executa *re-render* ao elemento em causa, não fazendo *re-render* aos filhos do mesmo.

- **Diretivas -** Contém diretivas embutidas que permitem manipular a DOM de maneira declarativa. As diretivas são escritas com o prefixo "**v-**"e são usadas para controlar a exibição, o *render* e o comportamento de elementos HTML.

- ***Data Binding* -** Oferece *data binding* **bidirecional**, o que significa que as alterações feitas nos dados do modelo são refletidas automaticamente na interface e vice-versa, tornando o desenvolvimento mais eficiente e reduzindo o risco de erros.

- ***Framework* Progressiva -** Pode ser usada de várias maneiras, dependendo das necessidades do programador, ou seja, é projetada para ser progressivamente adaptável. Isto significa que pode ser usada para adicionar recursos gradualmente a uma aplicação, permitindo uma maior flexibilidade e eficiência no desenvolvimento independentemente da complexidade do projeto.

- **Comunidade -** Possui uma comunidade ativa que cria e mantém uma grande variedade de *plugins*, ferramentas e recursos para auxiliar no desenvolvimento, tendo a maioria dos desafios mais comuns uma solução já consolidada.

2.3.3 *SvelteKit*

SvelteKit é uma *framework* relativamente recente, vocacionada para desenvolvimento *web* que tem ganho popularidade nos últimos tempos, funcionando em cima da *framework* ***Svelte***. Algumas das suas características incluem:

- **Código -** Ao contrário de certas *frameworks* como o *React* em que o código é interpretado em tempo de execução, esta apresenta uma estratégia como a compilação de código em *HTML* estático.

- **Funcionalidades -** Esta *framework* já possui várias funcionalidades básicas como *routing*, *server-side-rendering*, *build optimizations* e *preloading pages*.

No entanto, esta *framework* possui os seus pontos fracos como ser recente e não possuir uma grande comunidade de programadores, resultando numa escassez de documentação ou soluções para os mais variados problemas. Para além disto, não possui assim tanta bibliotecas como as restantes *frameworks* mais estabelecidas no mercado.

3 *Server-side Frameworks* Vs *Hybrid Frameworks*

3.1 *Frameworks Server-side*

As *frameworks* ***server-side*** são desenhadas para executar num servidor, o que significa que estas processam dados e lógica no servidor antes de enviar os resultados para o lado do cliente. São utilizadas para desenvolver aplicações *web* nas quais a maioria do processamento acontece no servidor e o lado do cliente apenas exibe os dados. Este tipo de *frameworks* é tipicamente usado para construir aplicações e serviços *web* mais complexos, ou seja, que requerem processamento de dados complexo, interações com bases de dados e *scripting* no lado do servidor. Algumas das vantagens de *frameworks server-side* incluem:

- **Performance -** Como todo o processamento ocorre no servidor, as aplicações são capazes de lidar com grandes quantidades de dados e tráfego sem afetar a experiência do utilizador.
- **Segurança -** Uma vez que todo o processamento ocorre no servidor, os programadores podem implementar medidas de segurança mais rigorosas para proteger os dados do utilizador.
- **Manutenção -** Possuem, geralmente, um conjunto de recursos que facilitam a manutenção de aplicações *web*, como ferramentas de atualizações de segurança.
- **Compatibilidade -** As aplicações *server-side* podem ser executadas em diferentes dispositivos e sistemas operacionais, uma vez que não dependem do navegador do utilizador.

Alguns exemplos deste tipo de *framework* incluem: **Ruby on Rails, Django** e **Node.js.**

3.2 *Frameworks* Híbridas

Por outro lado, as *frameworks* **híbridas** são concebidas para criar aplicações móveis que podem ser executadas em várias plataformas, como *Android*, *iOS* e *web*. Estas *frameworks* utilizam tecnologias *web*, como *HTML*, *CSS* e *JavaScript*, para desenvolver uma base de código única que pode ser compilada numa aplicação nativa para as diferentes plataformas. Algumas das vantagens deste tipo de *frameworks* incluem:

- **Reutilização de Código -** Permitem que os programadores reutilizem o mesmo código para criar aplicações *web* e móveis, o que pode economizar tempo e esforço.
- **Multiplataforma -** Podem ser executadas em diferentes sistemas operacionais móveis (como *iOS* e *Android*), bem como em navegadores *web*.
- **Acesso a Recursos do Dispositivo -** Podem aceder a recursos nativos do dispositivo, como a câmara e o GPS, o que pode oferecer recursos avançados para os utilizadores.
- **Custo -** A criação de aplicações híbridas pode ser mais económica do que a criação de aplicações nativas para diferentes plataformas móveis, pois requer menos esforço de desenvolvimento.

Alguns exemplos deste tipo de *framework* incluem: **Angular, React** e **Vue.js.**

4 Proposta de Arquitetura

Após a análise e a apresentação das várias *frameworks* possíveis para cada camada no desenvolvimento de aplicações, o nosso grupo irá apresentar uma arquitetura exemplo que poderá surgir a partir das *frameworks* discutidas.

Visto que as arquiteturas implementadas no mundo real são dependentes dos requisitos das aplicações a serem desenvolvidas, começamos pela definição de algumas metas a atingir e do foco da nossa arquitetura exemplo. Ainda no contexto de mundo real, as aplicações desenvolvidas são maioritariamente definidas por clientes e necessitam, frequentemente, de ser eficientes, escaláveis e de fácil manutenção. Deve-se também considerar outras necessidades como a capacidade de evolução da arquitetura perante novos requisitos do cliente, assim como a sua adaptação a novas tecnologias.

4.1 Camada de Dados

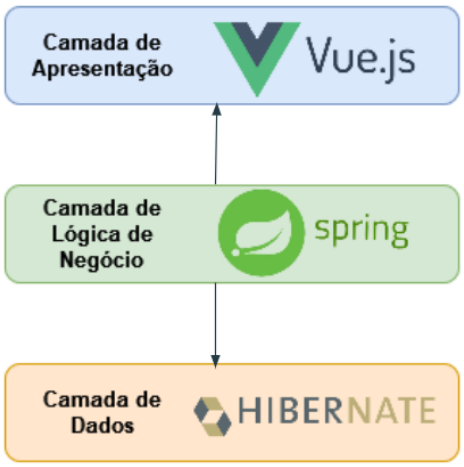
Para a camada de dados optamos por selecionar a *framework* **Hibernate**, pela sua rápida performance, proveniente da sua utilização interna de *cache* para evitar acessos repetidos, assim como a sua facilidade de manutenção devido às ferramentas fornecidas de consulta de estatísticas da base de dados.

4.2 Camada de Lógica de Negócio

Na lógica de negócio, utilizamos a *framework* **Spring**, baseada em *Java*, devido à sua modularidade e à sua flexibilidade, o que permite escalar a aplicação, assim como o seu relativo *Light Weight* e as suas poderosas capacidade na construção de aplicações empresariais, fornecendo também métodos de teste e de correção de problemas.

4.3 Camada de Apresentação

Para esta camada, recorremos à *framework* **Vue** devido à sua versatilidade e por se tratar de uma *framework* progressiva, sendo assim possível a utilização de diversos níveis de integração da mesma. Para além disto, devido à sua programação por componentes, conseguimos desenvolver aplicações modulares e, por isso, de fácil manutenção e reutilização.



5 Conclusão

Ao longo deste trabalho, foi possível explorar a importância das *frameworks* para as várias camadas aplicacionais, desde a camada de apresentação até a camada de dados.

As *frameworks* são ferramentas que permitem aos programadores criar aplicações de forma mais rápida, eficiente e com maior qualidade. Na camada de apresentação, estas permitem a criação de interfaces gráficas e a interação do utilizador com a aplicação de forma simples e intuitiva. Já na camada de negócio, as *frameworks* fornecem uma base sólida para a implementação da lógica de negócio, tornando o desenvolvimento mais ágil e consistente. E, finalmente, na camada de dados, oferecem ferramentas para uma interação segura e eficiente com as bases de dados.

Além disso, é de notar que a escolha da *framework* correta para cada camada aplicacional é fundamental para o sucesso do projeto. É necessário ter em consideração fatores como a finalidade da aplicação, a linguagem de programação utilizada, a usabilidade, a comunidade de suporte, a documentação e a flexibilidade da *framework*.