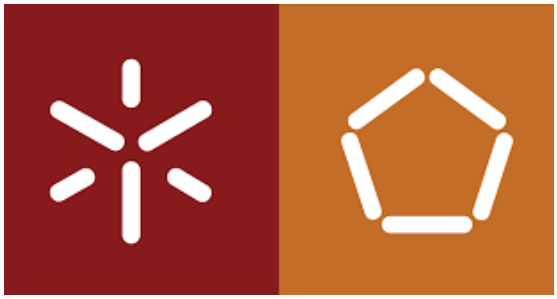
Universidade do Minho
Escola de Engenharia



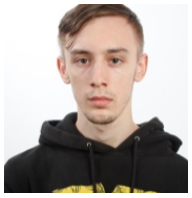
Arquiteturas Aplicacionais
Mestrado em Engenharia Informática

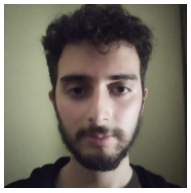
Pesquisa sobre *Frameworks* de separação de camadas

André Lucena Carlos Machado
Gonçalo Sousa João Castro José Barbosa


PG52672


PG52675


PG52682


PG53929


PG52689

21 de fevereiro de 2024

1 Introduction

Neste primeiro exercício, referente à unidade curricular de Arquiteturas Aplicacionais, fomos desafiados a realizar uma pesquisa sobre frameworks de separação de camadas. Para tal, e como proposto, procedemos à identificação de biblioteca e frameworks para cada camada do desenvolvimento do *software*, tocando nas vantagens e nas diferenças entre cada uma. Também realizamos uma breve discussão sobre frameworks, server side e frameworks híbridas, examinando as suas vantagens, desvantagens e os casos onde são mais adequados. E por último, apresentaremos uma proposta de uma arquitetura típica com base naquilo que estudamos.

1.1 Conceitos

No contexto das *frameworks*, há alguns conceitos que necessitam de desambiguação para melhor comunicação das suas características em relação uns aos outros.

Fortemente ou Fracamente Opinionada

Uma *framework* diz-se Fortemente Opinionada quando tem claramente uma forma correta de ser utilizada, com propósitos definidos e estrutura desenhada para os satisfazer, o que pode ser limitante para o desenvolvedor se tencionar quebrar as suas definições, no limite impossibilitando-o com erros de poder obter certos resultados na sua solução. Estas limitações são naturais no contexto de uma estrutura que tenha mais funcionalidades já convencionadas, como é o caso de uma *Template Engine* de se recusar a invocar funções do utilizador para evitar HTML cru.

Baixo ou Alto Acoplamento

Uma *framework* que proporcione baixo acoplamento serve-o possibilitando que as suas funcionalidades tenham poucos pontos de mudança e interação entre si, a linguagem nativa e mesmo as componentes que a integram. Esta é uma boa métrica no contexto da manutenção, por diminuir os pontos de contacto que necessitem de alteração entre diferentes módulos, tornando-a mais expedita e eficiente.

2 Presentation

Antes de analisarmos detalhadamente as possíveis tecnologias usadas na camada de Apresentação, seria importante observarmos que tipo de ferramentas são mais frequentemente usadas no mercado de trabalho nestes últimos anos, e como se tem comportado a variação da popularidade de cada uma destas.

Como podemos observar de seguida (Figura 1), através de um estudo realizado entre Nov-2022 a Dez-2023, num universo de 500,000 propostas de oferta de emprego, podemos averiguar que **React Native** é uma das *frameworks* mais solicitadas contando sensivelmente com metade da demanda de todo o universo em que foi realizado o estudo. Em segundo lugar, temos a *framework* **Angular** que conta com aproximadamente 173,000 ofertas. Em terceiro lugar, com cerca de 48,000 trabalhos, temos o **Vue**. Por fim, a restante oferta conta com *frameworks* como, **Svelte, Preact, Ember**, entre outros.

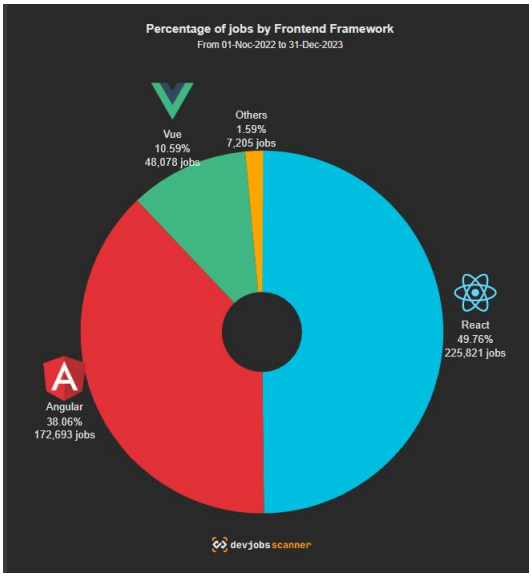


Figura 1: Percentagem da oferta em função da framework de frontend

No que toca em termos de popularidade, o comportamento é muito semelhante ao das ofertas mencionadas anteriormente.

2.1 Server Side Frameworks

Spring MVC - Java

O **MVC** é uma vertente do *Spring*, que permite implementar uma aplicação através da arquitetura *Model-View-Controller*. A *framework* oferece recursos específicos para o desenvolvimento de aplicações *web* como *routing* de pedidos.

Traz como principais vantagens:

- Altamente modular, desta forma os programadores podemos escolher apenas os componentes necessários para o projeto, o que torna o código mais conciso e com melhor manutenibilidade;

- Como segue o padrão *MVC*, há uma melhoria na organização do código e manutenibilidade;
- Consistência em termos de configuração, o uso de anotações permite um código mais consistente e menos confuso;
- É leve, pois usa um *servlet container* leve para desenvolver e fazer *deploy* da aplicação.

Struts - Java

Da mesma forma que o *Spring MVC*, o **Struts** segue o padrão arquitetural *Model-View-Controller (MVC)*.

Tem como pontos fortes:

- Como segue o padrão *MVC*, há uma melhoria na organização do código e manutenibilidade.
- Covenções em vez de Configurações, a *framework* assenta-se sobre convenções em vez de configurações, desta forma, podemos reduzir o nosso tempo com configurações e proceder diretamente para o desenvolvimento da aplicação.
- Suporte da comunidade, o **Struts** tem uma comunidade ativa, o que contribui com novos *plugins*, recursos e extensões.
- Customização, podemos adicionar *tag*, módulos e *plugins*.

2.2 Hybrid Frameworks

React - JavaScript

- Flexível, pois podemos conjugar *HTML* e *CSS* com *Javascript*. (JSX)
- Reutilização de código, pois permite a utilização de componentes já desenvolvidos.
- Eficiente, pois renderiza apenas os componentes que se modificam através da comparação do DOM do estado anterior.
- Grande variedade de ferramentas e bibliotecas.
- Aumento de complexidade assim que o projeto cresce.

Vue - JavaScript

- À semelhança do React permite a reutilização de código de componentes já desenvolvidos.
- Fácil de integrar com projetos.

- Documentação mais extensa e atualizada comparativamente com o React.
- Os dados são automaticamente atualizados quando ocorrem alterações devido ao sistema de *getters* e *setters*. No React é necessário usar o *hook* *useState* para gerir e atualizar o estado.
- Desenvolvimento ágil.

Angular - JavaScript

- É um *framework* robusto, pois integra recursos como gestão de estado, roteamento, um sistema de injeção de dependência, diretivas estendidas do HTML, entre outros.
- Possui uma estrutura organizada, seguindo o modelo MVC e MVVM.
- À semelhança do Vue oferece vinculação de dados bidirecional, ou seja, alterações que ocorram no modelo de dados são atualizadas automaticamente na *view*.
- A *engine* de *rendering* é complexa pelo que a *performance* de carregamento é mais baixa.

2.3 Bibliotecas pertinentes para *frontend*

Pretendemos nesta secção, para além do esforço do trabalho pedido, apresentar algumas bibliotecas existentes para a linguagem JavaScript.

- **PrimeVue**: biblioteca que fornece um conjunto de componentes para aplicacões Vue.
- **Material-UI**: biblioteca que fornece um conjunto de componentes de React (ajuda a construir uma interface mais moderna).
- **JQuery**: manipular e gerir eventos de elementos do DOM.
- **D3.js**: interagir com representações gráficas de dados.
- **Anime.js**: criação de animações.

3 Business Layer

3.1 Server Side Frameworks

Spring - Java

Inicialmente uma ferramenta de *dependency injection* o *Spring* é agora uma das *frameworks* mais populares no que toca ao desenvolvimento de *backend* sendo a sua utilização possível com *Kotlin* e *Groovy* para além de *Java*

- Oferece o padrão de *design Dependency Injection*, permitindo acoplamento fraco dos componentes e facilitando os testes à aplicação.
- Configuração flexível que oferece apoio para tarefas genéricas como conexão a bases de dados e tratamento de exceções.
- Comunidade ativa e documentação extensiva.
- **Spring boot**: que oferece uma abordagem opinionada utilizando o seu próprio julgamento para definir dependências iniciais, que valores por defeito utilizar com base nas necessidades do projeto.

Express/Node - JavaScript

O *Node* é um ambiente open-source que permite criar ferramentas e aplicações server-side em JavaScript. O *Node*, por si só, possui várias vantagens:

- O *Node* tem bom desempenho, uma vez que otimiza a taxa de transferência e a escalabilidade em aplicações *web*.
- O *node package manager* fornece acesso a centenas de milhares de pacotes reutilizáveis. Também é bastante capaz de resolver dependências e de automatizar maioria do *build toolchain* (em JavaScript consiste de *package manager*, *bundler* e *task runner*).
- O Node.js é portátil, estando disponível em Windows, Linux, macOS e outros mais sistemas operativos.

O *Express* é o *framework web Node* mais popular e é a biblioteca subjacente a vários outros *frameworks web Node*, sendo que é uma maneira simples e eficiente de construir aplicações web. Permite a definição de rotas para manipular diferentes solicitações HTTP, integração com mecanismo de renderização de visualizações e configurar parâmetros comuns de aplicações, como a porta para conexão e localização de modelos usados para renderizar uma resposta.

Trailblazer - Ruby

Uma *framework* para **Ruby** para estruturar a lógica de negócio através do objeto de serviço canónico *operation*, simplificando-a . Esta *framework* possui vantagens como:

- Diversas ferramentas de abstração que se baseam no objeto de serviço e no organizar do código em pequenos e diferentes *steps*.
- Erros lidados autmoaticaments utilizando a lógica de *railway* sem condicionais, que aumenta a mecânicas de herança e composição.
- Simplificar os Controladores e os Modelos no contexto *MVC* para serem o mais simples possível, retirando toda a lógica para dentro do serviço das *Operations*.

- Não se limita a oferecer *gems* para as *operations*, incorporando objetos de forma, componentes de view, entre outros.
- Agnóstico de outras *frameworks* escolhidas, como é o caso de *Rails*, incorporando facilmente diferentes necessidades.

3.2 Hybrid Frameworks

O Next.js pode ser uma boa opção no que toca a *frameworks* híbridas, através desta é possível proceder à renderização do lado do cliente e também do servidor. Com o Next.js é possível fazer o *rendering* de componentes em React do lado do servidor de forma a obter uma melhor *performance* do carregamento da página do lado do cliente. Por outro lado, assim que a página é carregada e os *bundles* são executados (js,css,bibliotecas externas, entre outros recursos), as interações seguintes apenas envolvem atualizações dinâmicas na DOM do lado do cliente, não sendo necessário "sobrecarregar"o servidor proporcionando uma experiência de utilizador mais fluída. O Next.js também oferece geração de páginas estáticas em *build time* fornecendo bom desempenho de carregamento, já que estas páginas não necessitam de dados dinâmicos.

4 Integration Layer

Todas as *frameworks* apresentadas são utilizadas no contexto na Linguagem *Java*.

4.1 Server Side Frameworks

JDBC

A *Java Database Connectivity* é uma *API* para a linguagem *Java* que define a conexão de um cliente a uma base de dados relacional.

É uma ótima opção se:

- Não pretendemos aprender nenhuma *framework*;
- Pretendemos um código simples;
- Queremos *queries* personalizadas;
- Uma solução *lightweight*, uma vez que permite escrever *queries SQL* diretamente invés de usar alguma abstração que trará uma maior peso na aplicação como consequência.

Ainda no contexto da **JDBC**, existem alternativas para reduzir significativamente o código *boilerplate* como *Spring JDBC template* ou *Apache DBUtils*, desta forma escrevemos menos linhas de códigos repetitivas e evitamos casos em nos esquecemos de alguma parte fundamental do processo de acessar os dados à base de dados como fechar à conexão, deixando um potencial erro futuro caso o limite de conexões/cursores à base dados seja atingido.

Hibernate

Entrando no domínio das *frameworks Object relation mapping* (ORM), entre elas a *framework open source* **Hibernate** é a mais popular. Tendo como foco abstrair o processo de criar *queries* e mapear as tabelas em objetos *Java* ao custo da escrita de anotações nas classes *Java*. Desse modo, conseguimos uma maneira mais neutra de comunicar com diferentes bases de dados relacionais e, ao mesmo tempo, reduzimos código *boilerplate*.

Sendo assim o **Hibernate** oferece como vantagens:

- Acesso simples aos dados, com o mapeamento objeto-tabela foi removida a necessidade de *queries* manuais uma vez que podemos interagir diretamente com objetos *Java*.
- *Open-source*, portanto, não é preciso nenhuma licença para utilizar o **Hibernate**.
- Suporta diferentes bases de dados, grande parte do código produzido não está diretamente ligado a uma base de dados em específico e, assim, trocar de base de dados é pouco dispendioso. O **Hibernate** possui uma linguagem orientada a objetos própria *HQL* com sintaxe semelhante à *SQL*, sendo convertida posteriormente na linguagem da base de dados correspondente.
- Aumenta a produtividade, ao remover código *boilerplate*, os programadores podem gastar mais tempo a melhorar a parte lógica da aplicação.
- *Cache*, o **Hibernate** suporta mecanismos de *cache* na forma de sessões, assim, mantém objetos em *cache*, o que diminui o número de chamadas à base de dados.

MyBatis

O **MyBatis** é uma alternativa intermédia ao **JDBC** e o **Hibernate**, pois abstrai grande parte do código do **JDBC** e ainda fornece uma maior liberdade comparado ao **Hibernate**. Tendo como pontos fortes:

- Controlo sobre as *Queries*, deste modo podemos otimizar *queries* mais complexas;
- Facilidade de uso para quem está habituado a *queries SQL*;
- Flexibilidade, consegue lidar bem com bases de dados não normalizadas.

5 Arquitetura Tipo

Tendo em consideração as *frameworks* apresentadas anteriormente, decidimos construir uma *stack* com as seguintes camadas, com a aplicação fundamentalmente baseada na linguagem Java:

- **Presentation**: *Vue.js(Client) + Spring MVC(Server)*
- **Business**: *Spring*
- **Integration**: *Hibernate*
- **Database**: *PostgreSQL*

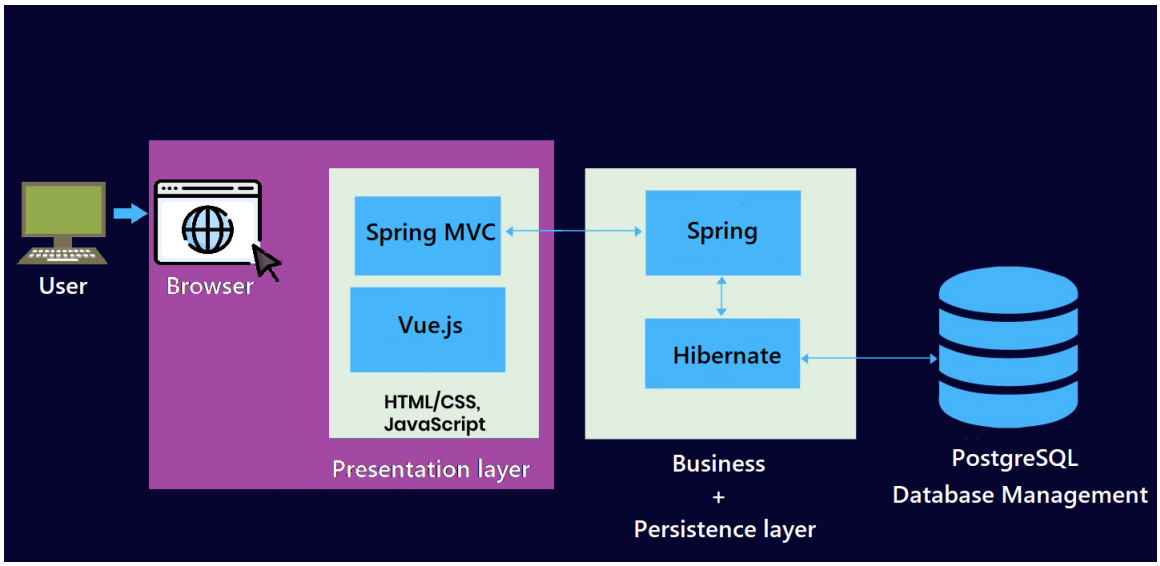


Figura 2: Arquitetura base

6 Referências

6.1 Presentation Layer

Referências React, Angular e Vue

- https://www.devjobsscanner.com/blog/the-most-demanded-frontend-frameworks/
- https://stackdiary.com/front-end-frameworks/
- https://rollbar.com/blog/most-popular-java-web-frameworks/#
- https://medium.com/vaadin/comparing-frontend-frameworks-for-spring-boot-react-angular-

Referências frameworks híbridas - Next.js

- https://stackshare.io/stackups/next-js-vs-nodejs

Referências Spring MVC e Struts

Referências sobre o **Spring MVC**:

- https://spring.io/projects/spring-framework
- https://spring.io/projects/spring-boot
- https://www.ibm.com/topics/java-spring-boot
- https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html
- https://www.geeksforgeeks.org/spring-mvc-framework/
- https://medium.com/@mohamed.enn/spring-boot-vs-spring-mvc-6a7981e46062
- https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm
- https://www.baeldung.com/spring-mvc-tutorial
- https://www.javatpoint.com/spring-mvc-tutorial
- https://www.javatpoint.com/java-spring-pros-and-cons

Referências sobre o **Struts**:

- https://struts.apache.org/birdseye.html
- https://en.wikipedia.org/wiki/Apache_Struts_2
- https://www.indiastudychannel.com/resources/148944-Pros-and-cons-of-Struts.aspx

Ainda relacionado com **Spring MVC e Struts**:
https://anywhere.epam.com/en/blog/java-spring-vs-struts

6.2 Business Logic Layer

Trailblazer

- https://www.codementor.io/ruby-on-rails/tutorial/trailblazer-office-hours-a-new-archit
- https://trailblazer.to/2.1/#some-code
- https://github.com/trailblazer
- https://www.youtube.com/watch?v=vgQMgIfVB00&t=0s

6.3 Integration Layer

É importante destacar que o processo de pesquisa debruçou-se muito sobre relacionar a (não muita) experiência com as bibliotecas/*frameworks* infracitadas com diversos *sites* e vídeos para minimizar ao máximo incoerências e até mesmo falsas verdades sobre o conteúdo.

JDBC

- https://www.ibm.com/docs/en/informix-servers/12.10?topic=started-what-is-jdbc
- https://en.wikipedia.org/wiki/Java_Database_Connectivity
- https://www.baeldung.com/java-jdbc
- https://www.baeldung.com/spring-jdbc-jdbctemplate

Hibernate, JPA e ideias sobre ORM's

- https://hibernate.org/orm/
- https://en.wikipedia.org/wiki/Hibernate_(framework)
- https://en.wikipedia.org/wiki/Jakarta_Persistence
- https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapp
- https://www.onlinetutorialspoint.com/hibernate/top-10-advantages-of-hibernate.html#google_vignette
- https://www.java4s.com/hibernate/main-advantage-and-disadvantages-of-hibernates/#google_vignette

Mybatis

- https://github.com/mybatis/mybatis-3
- https://www.tutorialspoint.com/mybatis/mybatis_overview.htm
- https://en.wikipedia.org/wiki/MyBatis

Informação útil sobre Comparações entre frameworks

Vídeos que fornecem uma introdução às *frameworks Java*:

- https://www.youtube.com/watch?v=XuLUnTlAWmw
- https://www.youtube.com/watch?v=1RQyNZbu8GY

Restantes comparações:

- https://www.baeldung.com/jpa-vs-jdbc
- https://www.baeldung.com/jpql-hql-criteria-query
- https://andreibaptista.medium.com/choosing-between-hibernate-and-mybatis-a-guide-to-ja
- https://www.reddit.com/r/learnjava/comments/ctk2ix/question_mybatis_vs_jdbc_template/
- https://topic.alibabacloud.com/a/first-the-difference-between-mybatis-and-jdbc_8_8_30006877.html
- https://www.quora.com/Why-is-MyBatis-preferable-over-plain-JDBC-in-larger-database-pro

6.4 Referências sobre Arquiteturas e Rendering

Para além dos slides de apresentação, procuramos elucidar o nosso conhecimento sobre a arquitetura das três camadas.

- https://www.linkedin.com/pulse/4-layers-software-application-explain-non-tech-linh-tra
- https://medium.com/@asoldan1459/three-layered-architecture-with-example-b597a2161538

Alguns apontadores sobre Rendering (SSR vs CSR vs Híbrido):

- https://clockwise.software/blog/client-side-vs-server-side-vs-pre-rendering/
- https://www.searchenginejournal.com/client-side-vs-server-side/482574/