Arquiteturas Aplicacionais

Uma imagem com ClipArt

Descrição gerada automaticamente

Exercício nº1

1 de março de 2021

**Grupo**

Filipa Alves dos Santos (A83631)

Hugo André Coelho Cardoso (A85006)

João da Cunha e Costa (A84775)

Luís Miguel Arieira Ramos (A83930)

Válter Ferreira Picas Carvalho (A84464)

Uma imagem com pessoa, parede, vestuário, sorriso

Descrição gerada automaticamenteUma imagem com pessoa, céu, homem, parede

Descrição gerada automaticamenteUma imagem com pessoa, parede, homem, interior

Descrição gerada automaticamente

Uma imagem com pessoa, parede, interior, jovem

Descrição gerada automaticamente

Mestrado Integrado em Engenharia Informática

Universidade do Minho

**Índice de conteúdos**

**1. Introdução 3**

**2. Desenvolvimento 4**

2.1. Bibliotecas vs. Frameworks 4

2.1.1. Inversão de Controlo 4

2.1.2. Extensibilidade e Código não modificável 6

2.2. Frameworks 7

2.2.1. Camada de Apresentação 7

2.2.1.1. Angular 7

2.2.1.2. Vue 7

2.2.1.3. Bootstrap 8

2.2.2. Camada de Lógica de Negócio 9

2.2.2.1. Express 9

2.2.2.2. Ruby on Rails 10

2.2.2.3. Spring 11

2.2.3. Camada de Dados 12

2.2.3.1. Hibernate 12

2.2.3.2. MyBatis 13

2.2.3.3. Mongoose 14

2.3. Tipo de Frameworks 15

2.4. Arquitetura 17

**3. Conclusão 19**

1. **Introdução**

O objetivo deste exercício foi fazer uma pesquisa sobre *frameworks* de separação de camadas, com a identificação de bibliotecas importantes e os pontos fortes de cada. Também foi pedido a diferenciação entre a *server side* e as híbridas (+*client side* ) bem como uma proposta de uma arquitetura em função das frameworks abordadas.

As *frameworks* são utilizadas para solucionar problemas recorrentes com uma abordagem comum. Com esta reusabilidade, facilita-se e acelera-se o desenvolvimento de *software* (desde que se conheça minimamente a *framework* em questão). Distinguem-se de librarias normais porque ditam o fluxo de controlo da aplicação, são extensíveis e não modificáveis, ou seja, o programador pode adicionar mais funcionalidades à *framework* mas não alterar o código da mesma.

Dentro das *frameworks*, podemos distinguir as que são exclusivamente *server-side*, isto é, as que apenas fornecem ferramentas para operações relacionadas com o servidor do *software*, e as híbridas, que para além disso têm a componente do cliente.

Neste relatório, é primeiro explicado a diferença entre bibliotecas e *frameworks* com maior detalhe*,* seguida de uma enumeração de diferentes *frameworks.* Para cada uma das camadas (apresentação, lógica de negócio e dados) são apresentadas 3 distintas, bem como os seus pontos fortes. Por fim, é brevemente analisado os diferentes tipo de *frameworks* e apresentada a arquitetura final sugerida pelo grupo.

1. **Desenvolvimento**

**2.1. Bibliotecas vs. Frameworks**

**2.1.1. Inversão de Controlo**

**BIBLIOTECA**

Quando se utiliza uma biblioteca, o fluxo da aplicação está no controlo do utilizador. Ou seja, é o utilizador que decide quando a utiliza e é apenas nesse momento que o código da biblioteca é executado. No exemplo do jQuery:

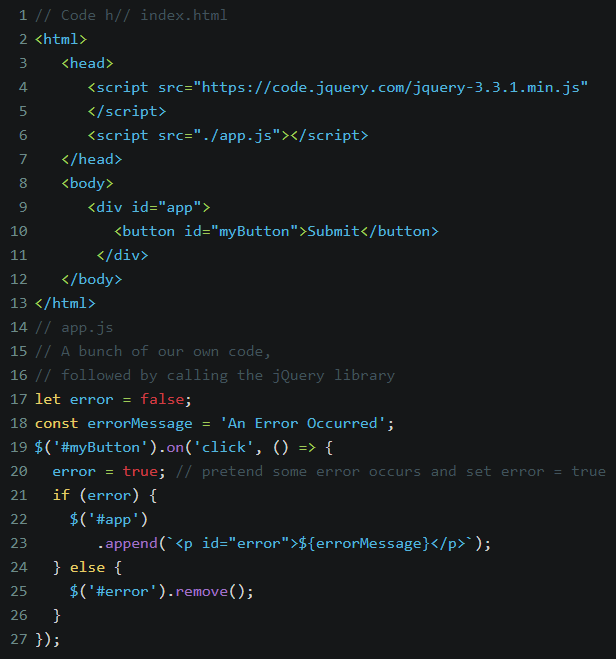


Figura 1 - Excerto de código HTML

Após análise do excerto de código seguinte, é possível concluir que o jQuery é uma biblioteca já que o utilizador decide onde e quando executar um determinado excerto de código, isto é, disponibiliza um determinado conjunto de funções e o utilizador decide concretamente quando as utilizar.

**FRAMEWORK**

Por outro lado, uma *framework* tem uma funcionalidade genérica que pode ser alterada utilizando código do utilizador, levando a que seja esta que esteja no controlo do fluxo do programa em que a única tarefa do utilizador fornecer os dados necessários à sua execução. Assim, são tipicamente chamadas "opinionated" visto que decidem localizações e nomes de ficheiros, eventos, escrita de código, entre outros. A grande vantagem de uma *framework* para um programador é evitar repetir configurações comuns a várias aplicações, poupando assim tempo que poderá ser orientado ao desenvolvimento da aplicação concreta. No exemplo do Vue.js:



Figura 2 - Excerto de código HTML 2

Analisando este excerto que tem o mesmo comportamento do anterior, conclui-se que o Vue.js é uma *framework*, uma vez que é o programador que utiliza construtores próprios fornecidos por esta, mas não está no controlo de quando nem como são executados.

**2.1.2. Extensibilidade e Código não modificável**

A extensibilidade das *frameworks* permite que sejam adicionadas novas funcionalidades a uma sem que o código base seja modificado. Normalmente, cada *framework* já é construída com este fator em mente, de modo a otimizar a sua funcionalidade.

No caso de uma libraria, estas são elaboradas para uma tarefa específica e a extensibilidade não é considerada como objetivo principal.

Além disso, a maior parte do código de uma *framework* é gerado e não pode ser alterado. Apesar de extensível, não é suposto que os programadores alterem o código já escrito.

**2.2. Frameworks**

Nesta secção iremos abordar três diferentes *frameworks* que decidimos apresentar, para cada uma das três diferentes camadas: camada de apresentação, camada de lógica de negócio e camada de dados.

**2.2.1. Camada de Apresentação**

**2.2.1.1. ANGULAR**

Angular é uma *framework* *open*-*source*, baseada em TypeScript, escrita parcialmente por uma equipa da Google. É uma *framework* de aplicações web e *front*-*end*.

**Vantagens**:

* **Two-way data binding**
  + automatiza a circulação de dados, não sendo necessário a criação de *handlers* para atualizar a componente View. Logo, quando o valor de um componente é alterado, o próprio *framework* atualiza a página, combinando a entrada e saída num único processo.
* **TypeScript**
  + o facto de TypeScript ser um *superset* de JavaScript e não uma linguagem *stand*-*alone* é uma mais valia pois permite encontrar erros mais facilmente (com o auxílio de um browser, p.e.) e facilita a manutenção de grandes aplicações.
  + adicionalmente, suporta tipos como primitivas, interfaces e outros (enums, etc).
* **Consistência, Reusabilidade e Produtividade**
  + a estrutura geral sistemática e a ferramenta CLI do Angular ajudam os desenvolvedores de um projeto a manter a sua consistência.
  + como consequência da consistência, os *developers* não precisam de perder tempo a tentar decifrar código, aumentando a produtividade.

**2.2.1.2. VUE**

Vue.js é uma *framework* JavaScript de código-aberto, focado no desenvolvimento de *user* *interfaces* e aplicações de página única (SPAs).

**Vantagens**:

* **Pequeno tamanho**
  + ferramenta que rápido download e instalação.
* **User-friendly e Simplicidade**
  + fácil utilização para novos programadores (só necessitam de saber JavaScript, HTML e CSS) e suportada por vários editores.
  + é possível obter bons resultados com poucas linhas de código, ideais para programas relativamente pequenos.
* **Fácil integração**
  + como é baseado em JavaScript, é fácil de integrar em outras aplicações já existentes – útil para novas aplicações web bem como alterar pré-existentes.
* **Flexibilidade**
  + a maneira de como o Vue foi construído oferece poucas restrições e uma maior flexibilidade para os programadores, mesmo que estejam habituados a utilizar React ou Angular. É possível escrever *templates* com, p.e., HTML e Javascript e adicionar ferramentas como saass (CSS Pre-processors) e pug (Template Engine).
* **Comunicação Bi-Direcional** 
  + semelhante ao Angular, devido à sua arquitetura MVVM (Model-View-Viewmodel), também integra “Two-way data binding”.

**2.2.1.3. BOOTSTRAP**

Bootstrap é uma *framework (open* *source*) dedicada ao desenvolvimento de interface para aplicações web. Utiliza CSS, HTML e JavaScript e é uma das *frameworks* *front*-*end* mais utilizadas hoje em dia.

**Vantagens**:

* **Flexível e Fácil utilização**
  + qualquer programador com algum conhecimento em HTML e CSS facilmente consegue usar Bootstrap e sua utilização acaba por lhe salvar muito tempo.
  + é também adaptável pois pode-se utilizar apenas as classes desejadas.
* **Compatível em diferentes browsers**
  + menos problemas de compatibilidade cross-browser
  + a página web ficará igual em todos os browsers modernos (como Firefox, Chrome, Internet Explorer, Opera etc).
* **Design Responsivo**
  + oferece um *grid* *system*  de 12 colunas que se adapta à resolução do dispositivo que utilizador está a visualizar a página web.
  + tem classes pré feitas para a *grid*, como sm, md e lg, que ajudam em definir a visibilidade de cada coluna ou componente, facilitando a construção de um site responsivo.
* **Updates regulares e Documentação útil**
  + o Bootstrap é das *frameworks* com um dos maiores números de *updates*, garantindo que se está a trabalhar com a versão mais recente das ferramentas disponíveis.
  + fornece ampla documentação, com ilustrações e demonstrações, para ajudar novos utilizadores com a aprendizagem. Também tem uma comunidade disposta a discutir possíveis problemas encontrados no caminho.

**2.2.2. Camada de Lógica de Negócio**

**2.2.2.1. EXPRESS**

Express é uma das frameworks mais populares para a construção de aplicações WEB em Node.js, pertencendo à camada de lógica de negócios.

**Vantagens:**

* **Desenvolvimento de aplicações facilitado e mais rápido por ser híbrido**
  + permite usar a mesma linguagem (JavaScript) tanto na *front*-*end* como na *back*-*end*, o que torna o processo mais fácil de gerir, visto que a mesma pessoa pode trabalhar com ambas as camadas de apresentação e de acesso a dados.
  + no *client*-*side*, permite a integração de motores de renderização de vistas, de forma a possibilitar a inserção de dados em templates (p.e. com PUG).
  + no *server*-*side*, permite o estabelecimento de definições comuns de aplicações web como a porta de ligação e a localização das templates renderizadas.
* **Tratamento de pedidos I/O**
  + ótima para aplicações que envolvem o tratamento de muitos pedidos HTTP na comunicação entre o *server*-*side* e o *client*-*side* – permite especificar *handlers* para vários inúmeros pedidos HTTP em rotas diferentes.
* **Comunidade *open*-*source***
  + o código desenvolvido é sempre analisado por outras pessoas e melhorado, além de que existe muita documentação compreensiva.
* **Integração fácil de *middlewares* e outros serviços *third*-*party***
  + possui uma elevada compatibilidade com uma extensa coleção de pacotes de *middleware* para diferentes problemas de desenvolvimento web, que podem ser integrados com grande liberdade ao longo da pipeline de execução.
  + alguns exemplos de ferramentas úteis que é possível integrar na aplicação com recurso a *middlewares* são o suporte de cookies, sessões, utilizadores, encriptação de dados, etc.
* **Fracamente opinativo**
  + não há uma única maneira considerada correta para resolver um dado problema. De facto, Express proporciona uma grande liberdade ao utilizador, sob a forma de um vasto leque de *middlewares* que ele pode decidir incorporar na aplicação, por praticamente qualquer ordem que pretenda, na pipeline de processamento dos pedidos. A aplicação pode ainda ser estruturada bastante arbitrariamente quanto ao número de ficheiros ou mesmo a estrutura de diretorias. Em suma, há poucas restrições no que toca à integração de serviços externos na aplicação, o que proporciona uma grande versatilidade à *framework*.

**Framework Híbrida:**

Express engloba tanto o lado do cliente como o lado do servidor, com a vantagem bastante relevante de usar a mesma linguagem, JavaScript, em ambos, o que facilita bastante a comunicação entre os dois e aumenta a legibilidade do código da aplicação como um todo.

**2.2.2.2. RUBY ON RAILS**

Ruby on Rails é outra *framework* muito popular de desenvolvimento de aplicações web, exclusivamente *server*-*side*.

**Vantagens:**

* **Fortemente opinativo e promove as melhores práticas de web development (consistente e time-efficient)**
  + ao contrário do Express, Ruby on Rails coloca a convenção antes da configuração - o seu objetivo é tornar o processo de desenvolvimento web fácil, previsível e bem estruturado. Como tal, a *framework* inclui já todas as bibliotecas e módulos necessários para o desenvolvimento de uma aplicação web e insere-se no paradigma MVC, fornecendo estruturas base para a base de dados, o serviço e a *front*-*end*.
  + encoraja a utilização de standards de web *development*, como HMTL, CSS e JavaScript para a interface do usuário e JSON ou MXL para a transferência de dados.
  + no fundo, limita a liberdade do utilizador no processo de estruturação da aplicação a troco de tornar o processo mais linear e simplificado.
* **Infraestrutura extensiva**
  + vai de encontro ao ponto anterior, RoR inclui um servidor web integrado e uma base de dados com geradores e scripts, o que poupa bastante trabalho ao utilizador na construção da aplicação através da modelação destes componentes.
* **Migração de dados eficiente**
  + o esquema usado para o armazenamento dos dados é agnóstico à base de dados, o que significa que é possível usar a aplicação RoR com múltiplos SGBD’s diferentes.
* **Escalável e seguro**
  + uma vez que a *framework* define toda a estrutura da aplicação, trata de tornar escalável e segura de usar. Isto pode ser uma complicação que surge em aplicações desenvolvidas do zero com outras *frameworks*, traduzindo-se numa grande quantidade de trabalho extra para o desenvolvedor.

**Framework Server-Side:**

Ruby on Rails é uma *framework* do lado do servidor, ou seja, tudo é processado e renderizado do lado do servidor e servido ao cliente. Sendo uma *framework* MVC, a parte mais dúbia desta definição são as vistas, uma vez que o modelo e controlador são ambos *server*-*side*. Contudo, o output standard de uma aplicação RoR usa uma DSL chamada Embedded Ruby (ERB) que é processada do lado do servidor, enviando posteriormente o HTML gerado para o browser. Logo, a aplicação faz tudo do lado do servidor.

**2.2.2.3. SPRING**

Spring é a *framework* pertencente à camada de negócios para desenvolvimento de aplicações no Java Enterprise mais popular, encapsulando vários módulos tipicamente usados tais como ORM, MVC, JDBC, entre outros.

**Vantagens:**

* **Compatível com vários dispositivos**
  + utiliza a JVM, ao escrever código nativo Java este é executável em qualquer dispositivo que o tenha instalado
* **Bastante modular** 
  + utiliza primariamente o MVC, tal como o Ruby on Rails
  + apesar de dispor imensas bibliotecas e classes, é possível selecionar apenas as necessárias ao desenvolvimento da aplicação e descartar as restantes. Para além disso, não obriga a criação de um servidor web, permite execução direta na JVM
* **Dispõe interface de manutenção de transações**
  + esta interface permite a manutenção de transações escaláveis de locais (base de dados singular) até globais (usando JTA, por exemplo)
* D**isponibiliza soluções opinativas** 
  + tal como o Ruby on Rails, o Spring dispõe soluções para além da MVC que colocam a convenção antes da configuração (Spring Boot e o Spring Roo), uma vez que um dos principais objetivos é ter uma aplicação Web inicial funcional e bem estruturada. Todas as configurações geradas apenas utilizam os módulos necessários de acordo com as necessidades do desenvolvedor da aplicação
  + encoraja a produtividade generalizando comportamento genérico reaproveitado por várias aplicações, tais como o acesso à camada de dados
  + não é necessário gerar código visto que o utilizador só necessita de editar ficheiros XML para alterar configurações

**Framework Server-Side:**

Spring é uma *framework* tipicamente *server*-*sided*, uma vez que segue o modelo MVC, todo os serviços RESTFUL são definidos no servidor assim como a geração de páginas, pelo que é entregue ao utilizador que realizou o pedido uma página já gerada no servidor. Assim, o Spring é uma *framework* totalmente *server*-*sided*.

**2.2.3. Camada de Dados**

**2.2.3.1. HIBERNATE**

O Hibernate é uma ferramenta *open*-*source* que fornece persistência a objetos relacionais e suporte a *queries* para qualquer aplicação de java. O Hibernate pode ser utilizado para bases de dados relacionais (Hibernate ORM) ou para bases de dados não-relacionais (Hibernate OGM).

Hibernate mapeia classes Java para tabelas na base de dados e tipos de dados Java para tipos de dados SQL aliviando o desenvolvedor das tarefas de programação relacionadas à persistência de dados mais comuns.

**Vantagens**:

* **Framework open source**
  + não tem qualquer custo associado.
* **Maior performance**
  + utiliza memória cache o que resulta num bom desempenho
* **Queries independentes da base de dados**
  + se existir uma mudança de base de dados não é necessário criar novas queries.
* **Independente da base de dados**
  + pode ser usado com vários tipos de base de dados.
* **Criação automática de tabelas**
  + fornece a capacidade de criar tabelas automaticamente, não havendo necessidade de as criar manualmente.
* **Minimiza o acesso à base de dados com estratégias de "smart fetching"**

**2.2.3.2. MYBATIS**

MyBatis surge como uma alternativa para JDBC e Hibernate (algumas das sugestões apresentadas anteriormente). Este automatiza o mapeamento entre base de dados SQL e objetos em Java, .NET e Ruby on Rails. Difere de outras *frameworks* ORM uma vez que não mapeia objetos Java em tabelas (presentes nas bases de dados), mas sim métodos Java para SQL *statements*.

Uma diferença significativa entre MyBatis e outras estruturas de persistência é que MyBatis enfatiza o uso de SQL, enquanto outras estruturas como Hibernate normalmente usam uma linguagem de consulta personalizada, ou seja, a linguagem de consulta Hibernate (HQL) ou Enterprise JavaBeans Query Language (EJB QL).

MyBatis deixa usar todas as funcionalidades das bases de dados como procedimentos, vistas ou *queries*. Tal como o Hibernate, possuí cache. Neste caso, baseada num HashMap Java.

**Vantagens:**

* **Simplicidade e desenvolvimento rápido**
* **Código aberto**
  + é gratuito e um software de código aberto.
* **Suporta procedimentos armazenados**
  + encapsula SQL na forma de procedimentos armazenados para que a lógica de negócios possa ser mantida fora da base de dados e a aplicação seja mais portátil e mais fácil de implementar e testar.
* **Portabilidade**
  + pode ser implementado para quase qualquer linguagem ou plataforma, como Java, Ruby e C # para Microsoft .NET.
* **Interfaces independentes**
  + fornece interfaces independentes de base de dados e APIs que ajudam o resto da aplicação a permanecer independente de quaisquer recursos relacionados à persistência. MyBatis integra-se com Spring Framework e Google Guice, permitindo construir código de negócios livre de dependências.
* **Suporta SQL embutido**
  + nenhum pré-compilador é necessário e é possível ter acesso total a todos os recursos do SQL.
* **Suporta SQL dinâmico**
  + fornece recursos para construção dinâmica de consultas SQL com base em parâmetros. As instruções SQL podem ser construídas dinamicamente usando uma linguagem embutida com sintaxe semelhante a XML ou com Apache Velocity usando o plugin de integração Velocity.

**2.2.3.3. MONGOOSE**

Mongoose é uma *framework* de Modelação de Dados de Objetos (ODM) para MongoDB e Node.js. Gere relações entre dados, fornece validação de esquemas, e é utilizada para traduzir entre objetos em código e a representação desses objetos no MongoDB.

**Vantagens:**

* **Schemas**
  + usa uma base de dados NoSQL, o que proporciona um modelo de dados mais flexível à medida que evolui ao longo do tempo.
* **Validação**
  + tem um sistema de validação para a definição de esquemas. Isto poupa o trabalho de escrever código de validação.
* **Resultados de retorno**
  + facilita a devolução de documentos atualizados ou resultados de consultas. Em vez de devolver um objeto com uma flag de sucesso e o número de documentos modificados, o Mongoose devolve o próprio objeto atualizado para que se possa trabalhar facilmente com os resultados.
* **Extensa documentação e comunidade**
  + tem uma comunidade muita ativa e diversos documentos de apoio ao que facilita a sua aprendizagem e utilização.

**2.3. Tipo de Frameworks**

Atualmente, as *frameworks* podem ser *server*-*side*, *client*-*side* ou híbridas. Numa solução ***client-side***, um pedido do utilizador é redirecionado para um ficheiro HTML. Enquanto é feito o download de todo o JavaScript/HTML, o servidor devolve ao *user* um *loading* *screen*. Por fim, o browser compila tudo e só depois dá *render* à resposta.

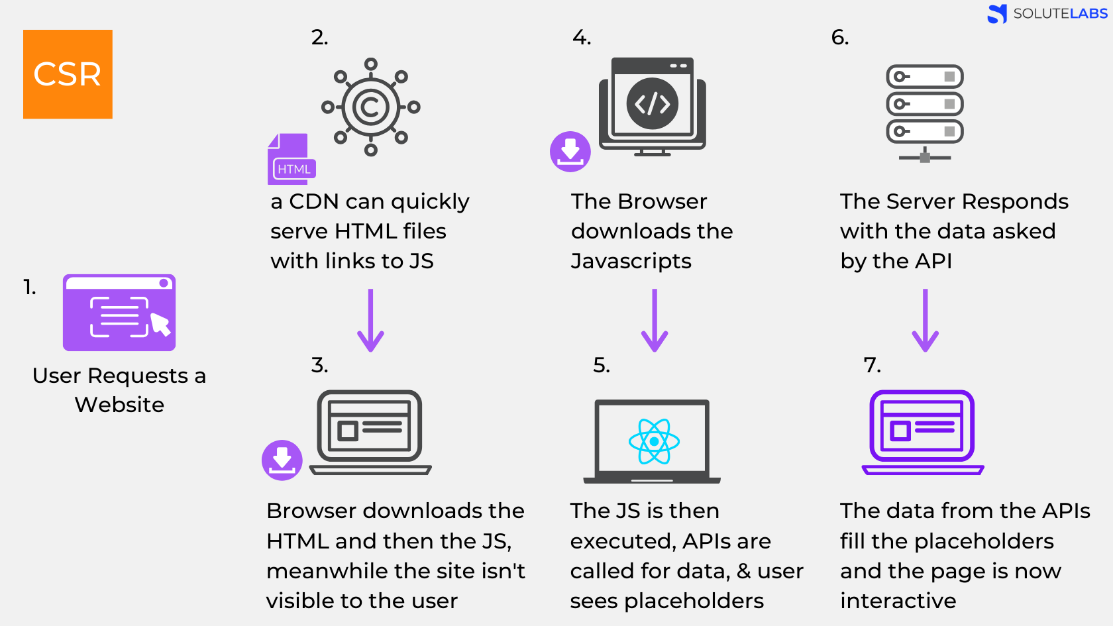


Figura 3 - Client-side rendering (CSR)

Já no caso de uma abordagem ***server*-*side***,o HTML é compilado antes de se efetuar o download do Javascript, o que faz com que a página seja apresentada antes ao cliente, mas só interativa quando JS é executado.

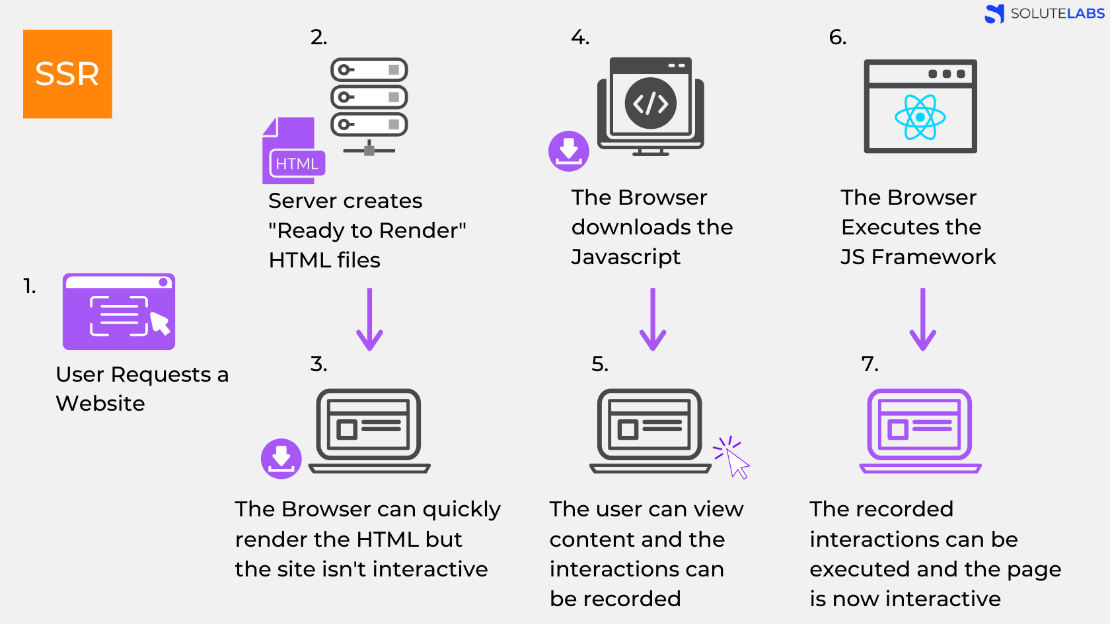


Figura 4- Server-side rendering (SSR)

Em termos de comparação, CSR perde mais **tempo** a dar *load* à primeira página relativamente a SSR. No entanto, nas seguintes páginas, já não necessita de tanto tempo pois, ao contrário de SSR, não necessita de repetir o ciclo todo. Quanto a **caching**, apesar de ambas as estratégias beneficiarem de tal, CSR tende a beneficiar mais, sendo que a partir do momento em que a aplicação está *loaded*, não é necessário fazer mais pedidos ao *server*. Em SSR, o pedido ao *server* é sempre mandado.

De modo a beneficiar de ambas as estratégias, temos uma abordagem **híbrida** que mistura ambas as técnicas de CSR e SSR.

**2.4. Arquitetura**

Para a arquitetura, decidimos tentar construir uma arquitetura orientada a eventos, procurando escolher *frameworks* que se complementassem e se adaptassem bem a um ambiente com muita interação I/O. O exemplo em mente ao longo deste processo foi um website.

A arquitetura orientada a eventos é usada sobretudo em sistemas com muita intervenção do utilizador, onde o programa passa a maior parte do tempo à espera de inputs – é assíncrono. Esta arquitetura é especialmente adequada a este paradigma porque se foca em criar uma unidade central que processa todos os inputs e os encaminha para os respetivos *handlers* distintos, isto é, processa um evento e produz o respetivo output, dependente do tipo de input.

Começando pela camada de dados, optamos por usar Mongoose, uma vez que possui um esquema não-relacional, o que é bastante vantajoso, uma vez que permite guardar dados de maneira menos rigorosa e proporciona uma grande escalabilidade – isto faz bastante sentido neste tipo de ambientes, por exemplo websites, em que é muito frequente ir criando novas funcionalidades ao longo do tempo, o que seria muito mais complicado com um esquema relacional rígido, visto que seria necessário reestruturar a base de dados a cada atualização que envolvesse um novo tipo de dados.

No que toca à camada de lógica de negócio, a *framework* de eleição foi o Express, uma vez que também é orientado a eventos. Esta *framework* executa um ciclo central que processa eventos e executa as *callbacks* correspondentes, fazendo o processamento assíncrono dos dados, que é ideal para websites, onde os inputs podem ser esporádicos e irregulares. Além disso, outra grande vantagem é a grande compatibilidade com Mongoose, uma vez que Express usa JavaScript, sendo assim possível realizar a tradução direta dos dados armazenados para objetos JSON. Além disso, o Express é bastante liberal quanto à estrutura da aplicação e disponibiliza um vasto leque de *middlewares* para incorporar na aplicação, o que também contribui para a escalabilidade do sistema.

Finalmente, para a camada de apresentação optamos por Vue. Esta *framework* permite definir handlers encapsulados de eventos, nomeadamente cliques de ratos ou de teclas, através de diretivas associadas a certos elementos da vista, garantindo o correto mapeamento dos eventos ao código respetivo. É uma *framework* reativa, pelo que consegue recarregar elementos individuais da *front*-*end* em resposta a eventos, evitando a necessidade de recarregar a página inteira, permitindo assim uma gestão mais eficiente e adequada a este paradigma. É também fácil estender a aplicação a novos tipos de eventos que surjam, sendo facilmente escalável. Além disso, Vue opera também em JavaScript, pelo que é bastante compatível com Express.

Concluindo, esta arquitetura é mais adequada a sistemas assíncronos com fluxo de dados imprevisível, sobretudo interfaces de utilizadores, com código bastante compartimentado e mapeado aos respetivos eventos e blocos de dados. A combinação de *frameworks* escolhida proporciona um processamento bastante eficiente e encapsulado dos eventos ao longo das camadas, recorrendo a um estado centralizado na camada de visualização para evitar acessos desnecessários à base de dados, bem como uma grande escalabilidade do sistema, dispondo de formas simples e relativamente independentes do resto do código de armazenar novos tipos de dados e criar novos eventos para os processar.

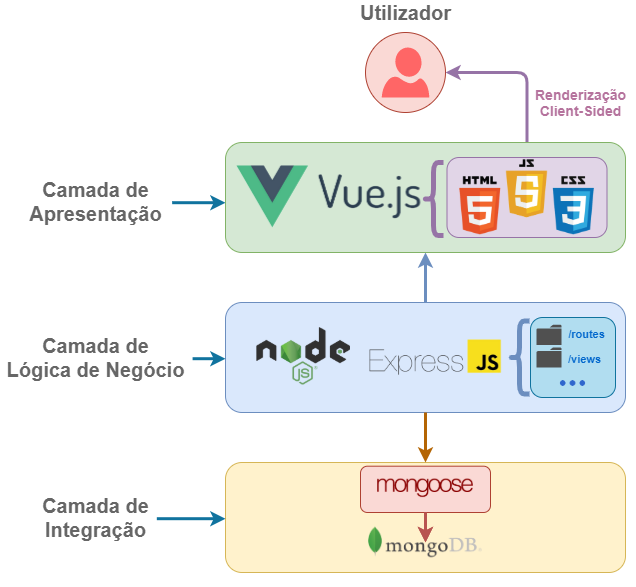


Figura 5 - Arquitetura sugerida

1. **Conclusão**

Finalizada a elaboração deste primeiro exercício prático, importa referir que a execução do mesmo permitiu aos elementos do grupo compreender a função e a importância das *frameworks* de separação de camadas.

A execução deste exercício prático permitiu uma melhor consolidação dos construtos teóricos através da análise e pesquisa sobre *frameworks*.

No ímpeto geral, o desenvolvimento deste exercício trabalho decorreu como planeado, alcançando os objetivos delineados pelo enunciado.