

ARQUITETURA DE REFERÊNCIA - GUIA FRONTEND

Versão: 1.2

Data	Versão	Descrição	Responsável
06/05/2019	1.0	Criação do Guia de Frontend da Arquitetura de Referência	José Augusto de Jesus
15/05/2019	1.1	Homologação do Guia Geral	Carlos Alberto Rodrigues Santana José Arthur Souza de Macedo José Augusto de Jesus Leonardo Moraes Borges Théo Alves Monteiro Valdecy Lourenço de Araújo Júnior
03/06/2019	1.2	Alteração do item 5.3 e Anexo B, incluindo os principais componentes CAPES.	Carlos Alberto Rodrigues Santana

SUMÁRIO

1.	INTRODUÇÃO.....	4
2.	PROPÓSITO	4
3.	ESCOPO	4
4.	REPRESENTAÇÃO ARQUITETURAL	4
4.1	<i>SERVICE ORIENTED FRONT END ARCHITECTURE – SOFEA</i>	8
5.	METAS E RESTRIÇÕES ARQUITETURAIS.....	10
5.1	CAMADA	11
5.1.1	MÓDULO (<i>MODULE</i>)	11
5.1.2	COMPONENTE (<i>COMPONENT</i>)	12
5.1.2.1	HTML (<i>TEMPLATE</i>)	12
5.1.2.2	SERVICE	12
5.1.2.3	FUNÇÕES.....	12
5.2	<i>FRONT-END</i>	12
5.3	COMPONENTE.....	13
6.	REQUISITOS DE QUALIDADE	14
7.	DEFINIÇÕES, ACRÔNIMOS E ABREVIACÕES	14
8.	REFERÊNCIAS	15
9.	ANEXO A – TECNOLOGIAS.....	16
10.	ANEXO B – COMPONENTES.....	16

1. Introdução

Este documento é uma especialização do **Guia Geral de Arquitetura** e tem natureza subsidiária ao Guia Geral.

2. Propósito

Apresentar arquitetura para o desenvolvimento e sustentação de *Front-end Web (Frontend)*. Detalhar os aspectos técnicos relativos à adoção de tecnologias, ao desenvolvimento e à implantação de componentes. Capturar e formalizar as principais decisões tomadas com relação à arquitetura de *Frontend*. Assim, não tem a intenção de ser um documento definitivo de arquitetura para cada sistema individual. Ao invés disso, tenta traçar os objetivos de alto nível que levem a uma boa arquitetura de software. Cada sistema em particular deve ser analisado junto com a equipe de desenvolvimento e um dos arquitetos de sistemas da CAPES. Se os tópicos aqui apresentados forem seguidos, as chances de sucesso nos projetos tendem ser maiores.

3. Escopo

Fornecer uma visão arquitetural usada no desenvolvimento de *Front-end*. Capturar e transmitir as decisões arquiteturais significativas que foram tomadas em relação ao desenvolvimento dos sistemas da CAPES.

Alinhar o desenvolvimento de *Front-end* a uma arquitetura baseada em componentes reutilizáveis, testáveis e com codificação dentro de padrões e guias de codificação de mercado.

Definir os parâmetros necessários a extensão deste documento.

4. Representação Arquitetural

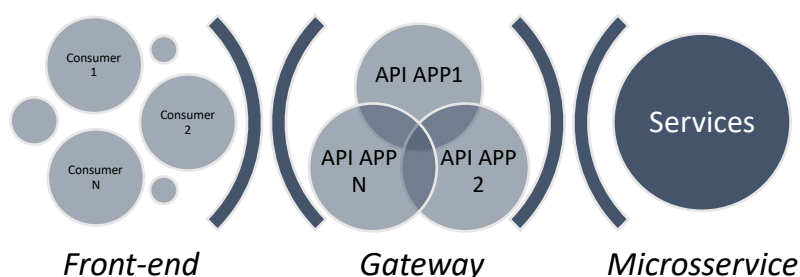


Figura 1 Fronteiras de responsabilidade das camadas

A *Figura 1* generaliza o padrão de projeto MVC, o qual, em sua essência, divide o desenvolvimento de software em três camadas: visão do cliente, controle de requisição e processamento de regra de negócio.

Objetivamente, este documento trata apenas da camada de visão do cliente (*View/User Interface*) e, devido à quantidade de *Framework* específico para ela, limita-se em definir premissas e restrições que devem ser atendidas na construção dessa camada.

O *Front-end* deverá ser funcionalmente independente do *Back-end*, mas dependente semanticamente.

Manter o isolamento de responsabilidade entre *Front-end* e *Back-end*. O grau de conformidade empregado entre *Front-end* e *Back-end* será definido pela API disponibilizada pelo serviço consumido.

A camada de *Front-end*, em direção ao *Back-end*, conhece apenas a *API* do serviço que deseja consumir conforme ilustrado na *Figura 1*.

Construir o *Front-end* sob a abordagem *Single Page Application* (SPA) empregando *template*, o qual deverá ser processado exclusivamente no lado do cliente.

Usar componentes para criar módulos e módulos para páginas. Cada componente deverá ser testável individualmente e em colaboração, quando compuser módulo. Em qualquer dos casos, *mock* ou *stub* deverão ser suficientes para que o componente funcione corretamente.

Priorizar o uso de *Dependency Injection* (Injeção de Dependências - DI) em detrimento de instâncias diretas no código de negócio. Nesse contexto, usar gerenciador de *DI* nativo do *Framework* em uso.

Construir componente estruturado por *template* HTML, o qual permita ter sua aparência especializada por folha de estilo própria sem que ela influencie nos demais componentes e tenha uma classe para gerenciamento de comportamento dos elementos que o compõe.

Codificar o *Front-end*, por conseguinte todos os seus componentes, usando TypeScript em sua última versão estável. Encapsular regras de negócio de *Front-end* em classe de serviço, a qual fará ligação entre o componente, *Front-end*, e o *Back-end*. A *Figura 2* ilustra o fluxo entre Componente, Serviço e *Back-end*.

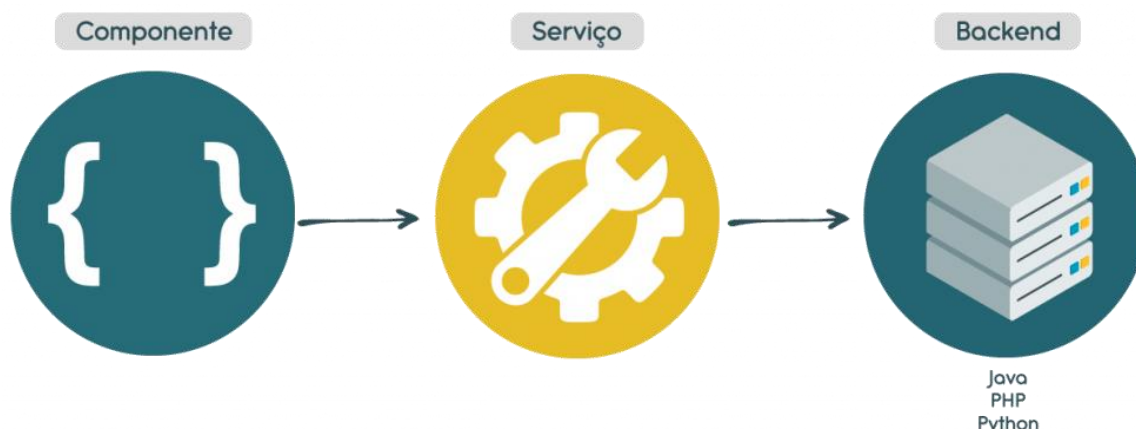


Figura 2 Comunicação entre componente, serviço e Back-end

Carregar a parte essencial do sistema para máquina do cliente. A partir desse ponto, cada requisição demanda apenas os dados que lhe são essencialmente necessários.

Realizar requisições ao *endpoint* através de sua API, via *Representational State Transfer* (REST), o qual retornará em formato *JavaScript Object Notation* (JSON) válido.

Empregar verbos HTTP para denotar o tipo de operação que se deseja executar contra um *endpoint*. A Tabela 1 elenca os verbos possíveis para realizar requisições ao *endpoint*.

Verbo HTTP	Descrição
POST (Create)	Criar recurso Alguma alteração poderá ser produzida por uma requisição PUT
GET (Read)	Recuperar informação de um recurso ou uma coleção deles Nenhuma alteração será produzida por uma requisição GET
PUT (Update/Replace)	Atualizar total ou parcialmente um ou mais recursos Substituir um ou mais recursos Alguma alteração poderá ser produzida por uma requisição PUT
DELETE (Delete)	Excluir um ou mais recursos Alguma alteração poderá ser produzida por uma requisição DELETE

Tabela 1 Emprego de verbos HTTP

A Tabela 1 tem o objetivo de correlacionar os verbos HTTP às operações. As especificidades requeridas pelo uso de cada verbo deverão ser observadas conforme definição no documento *REST API Best Practices*. **Erro! Indicador não definido..**

Usar código HTTP para interpretar o *status* da operação solicitada. Esta diretriz tem alinhamento direto com o *Back-end*, pois, é nesta camada que os códigos listados na Tabela 22 são definidos e retornados ao *Front-end*.

Código	HTTP	Objetivo
200	Ok	Informar que a requisição fora atendida adequadamente
204	Ok	Informar que a requisição fora atendida adequadamente, mas não retorna nenhum conteúdo
304	Not Modified	Informar que o recurso solicitado não sofreu modificação desde a última requisição
400	Bad request	Informar ocorrência de erro negocial
401	Unauthorized	Informar que o recurso solicitado requer autenticação
403	Forbidden	Informar que a credencial apresentada não permite o uso do recurso solicitado
404	Not found	Informar que o recurso solicitado não foi encontrado
412	Precondition failed	Informar erro de validação dos dados enviados
500	Internal Server Error	Informar erro não tratado no servidor

Tabela 2 Código HTTP usados pela API

O emprego de *REST* deverá observar as seguintes restrições, as quais estão sucintamente descritas na *Tabela 3*. A versão completa e detalhada de cada uma das restrições apresentadas na *Tabela 3*, poderá ser conferida no documento *REST API Best Practices*¹.

Princípio	Objetivo
<i>Uniform Interface</i>	Simplificar e desacoplar a arquitetura Evoluir partes da aplicação de forma independente
<i>Resource-Based</i>	Identificar recursos (serviço) univocamente Retornar ao cliente apenas o estritamente necessário Evitar desperdício de recurso de rede
<i>Manipulation of Resources Through Representations</i>	Permitir que o cliente tenha informações suficientes para modificar ou excluir um recurso no servidor, desde que possua permissão para tanto
<i>Self-descriptive Messages</i>	Incluir informações suficientes para descrever como processar a mensagem em cada mensagem
<i>Stateless</i>	Permitir extração de todas as informações de estado da própria requisição, as quais serão enviadas como parte da parte da URI
<i>Cacheable</i>	Indicar, explicitamente, a capacidade de <i>cacheamento ou não de um recurso</i>
<i>Client-server</i>	Definir interfaces uniformes entre <i>Cliente</i> e <i>Servidor</i> Separar responsabilidades de cliente no cliente e de servidor no servidor. Exemplo: Ao cliente, jamais, deverá ser atribuído a responsabilidade de armazenamento de dados e ao servidor, não cabe depender da versão de navegador para que sua regra de negócio funcione corretamente ²
<i>Layered system</i>	Limitar o conhecimento do cliente sobre o local real de armazenamento do recurso por ele requisitado. Cabe-lhe apenas saber que o recurso está disponível em um dado <i>endpoint</i> e não onde o endpoint está hospedado

Tabela 3 Restrições Arquiteturais do REST

¹ A versão completa do documento que detalha cada uma das restrições está disponível em:
<https://tinyurl.com/rest-espect>

² Salvo quando a própria regra de negócio versar sobre isso.

4.1 Service Oriented Front End Architecture – SOFEA

Todas as premissas e restrições apresentadas no item 4 formam a base para o desenvolvimento de software aplicando o paradigma, adotado pela CAPES, **SOFEA**.

Nesse modelo, logo na primeira requisição, todo o HTML, *templates*, JavaScript e CSS, **essenciais e necessários** à execução da aplicação são enviados para o cliente e, a partir daí requisições subsequentes e sob demanda são realizadas para complementar as funcionalidades da aplicação.

A Figura 3 ilustra de forma geral o processo de inicialização de uma aplicação sob a arquitetura SOFEA.

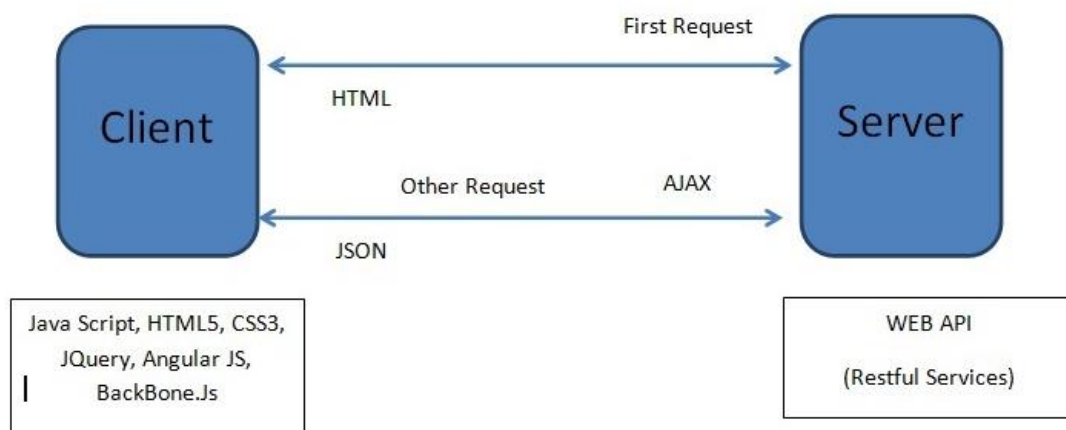


Figura 3 Visão geral de funcionamento de aplicação SOFEA

Uma das grandes dificuldades em se criar aplicações cliente/servidor web é gerenciar a expectativa do *download* inicial de grandes arquivos (HTML/CSS/JavaScript) necessários à inicialização da aplicação. A aplicação *Front-end* deve ter sempre em foco a experiência do usuário. Com isso, ter estratégias bem definidas para que o cliente seja capaz de interagir com o sistema o mais rápido possível.

Cada aplicação deverá ter um *bootstrap* mínimo e o menor possível para iniciar aplicação de modo a coloca-la operacional.

Deve atentar para o fato que, em uma primeira análise, aplicações em arquitetura SOFEA se confunde, em certa medida, com aplicações que utilizam o fluxo Ajax.

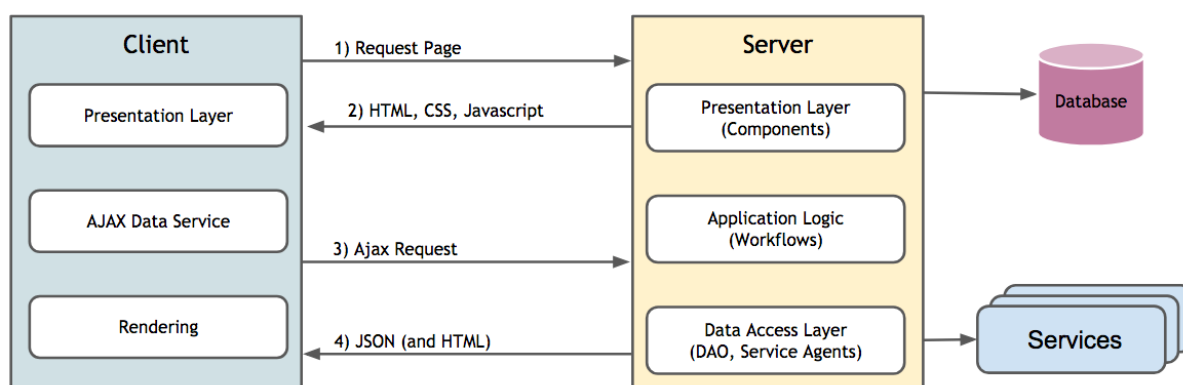


Figura 4 Arquitetura de aplicação web usando fluxo Ajax

Conforme observa-se na Figura 4, em uma aplicação clássica usando Ajax para baixar os *templates*, a camada de apresentação, responsável por gerenciar o que será mostrado ao usuário, fica parte no cliente e parte no servidor. Outro ponto é a camada que gerencia o fluxo de trabalho ou de eventos de tela (*workflow*), ela fica inteiramente no servidor e conforme vai sendo solicitada, o servidor faz o processamento da regra de negócio e também o de *workflow*.

Sob o prisma de aplicações SOFEA, também conhecida por *Thin Server Architecture*, preconiza que a lógica de renderização deverá ser exclusiva do cliente, eliminando do servidor a inteligência de geração de *templates* ou processamentos de *views*.

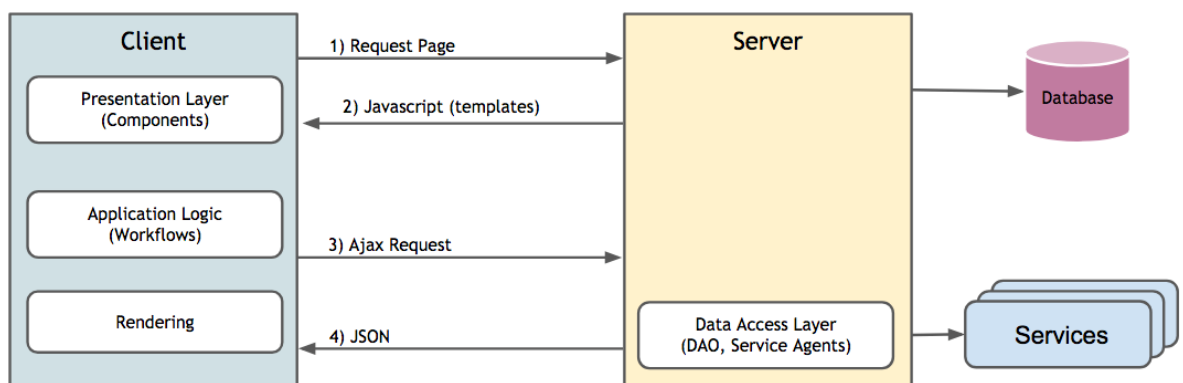


Figura 5 Arquitetura de aplicação Single-page Application

Com SOFEA, toda a responsabilidade de gerenciamento de apresentação fica do lado do cliente. Ela visa alcançar 3 objetivos principais:

- O programador Back-end pode focar na lógica de negócio da aplicação
- A aplicação torna-se menos complexa, uma vez que o *Back-end* e *Front-end* são separados
- Comunicação entre Cliente e Servidor usa um protocolo, sendo que cada um pode exportar, importar e apresentar dados para outro sistema (REST)

A arquitetura de aplicações em SOFEA permite em grande medida a separação de conteúdo estático do dinâmico. Isso também permite aliviar a carga dos servidores e diminuir o consumo de banda. Com essa diretriz em pauta, o Front-end deverá ser projetado para ter partes consumidas de diferentes servidores provedores de conteúdo (*Content Delivery Network – CDN*).

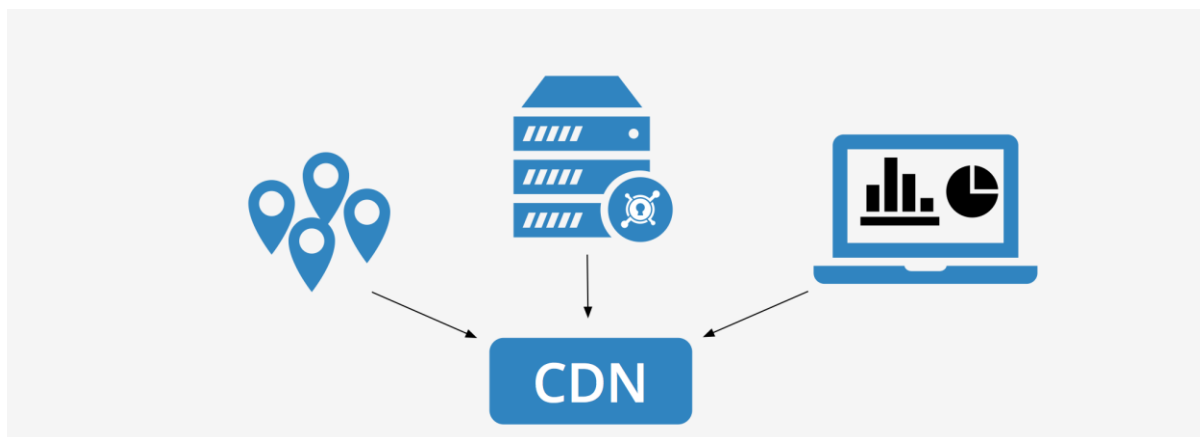


Figura 6 Arquitetura CDN

Nessa arquitetura, aplicações, clientes e serviços podem demandar conteúdo de uma CDN, basta que haja permissão suficiente para tal. Isto implica que o *Front-end* deverá prever mecanismos de consumo tanto de serviços de autenticação quanto identificar-se para outros serviços – saber gerir de *token* quando necessário.

5. Metas e Restrições Arquiteturais

Os indicadores de qualidade que cada sistema deverá apresentar ao ser submetido à avaliação por meio de ferramenta Sonarqube, compreende em cada indicador uma meta a ser alcançada. No que tange ao código-fonte, o quadro a seguir relaciona os indicadores e sua respectiva meta.

Grupo	Indicador	Tipo Meta	Meta
PROJETO	Problemas confirmados	Unidades	= 0
	Complexidade	Média total	<= 10
	Métodos	Média total	<= 3
	Índice de manutenibilidade	Nota	A
	Índice de confiabilidade	Nota	A
	Índice de segurança	Nota	A
	Taxa de dívida técnica	%	<= 2,5%
	Classes	Média total	<= 10
	Arquivos	Média total	<= 10
	Linhas duplicadas (%)	%	<= 4%
VIOLAÇÕES	Problemas impeditivos	Unidades	= 0
	Problemas críticos	Unidades	= 0
TESTE	Testes unitários ignorados	Unidades	= 0
	Sucesso em testes unitários (%)	%	= 100%
	Cobertura (camada de negócio)	%	>= 70%

Os projetos devem ser aderentes ao modelo de acessibilidade do governo eletrônico (EMAG), especialmente quanto a identificação dos campos e mensagens de ajuda para navegação na interface.

5.1 Camada

Os itens apresentados a seguir deverão ser entendidos como camadas, salvo direcionamento contrário, mesmo que em outras linguagens tenha entendimento diverso. Ainda, salvo menção explícita, sempre será referenciada apenas *Front-end* e embora sejam adotadas referências da arquitetura Angular, elas poderão ser reproduzidas por qualquer outro Framework desde que atendidas as especificações deste documento.

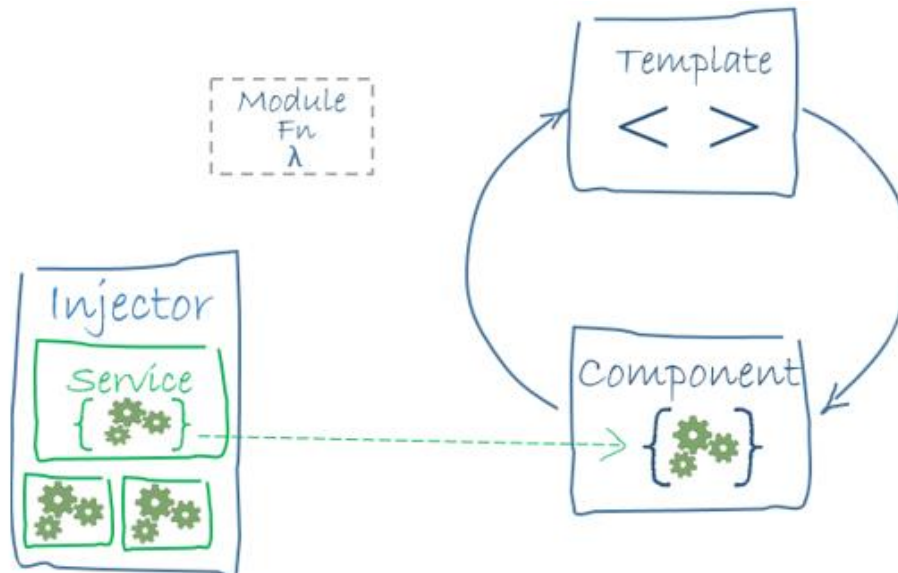


Figura 7 Arquitetura de componentes

5.1.1 Módulo (Module)

Agrupar e coordenar um conjunto de um ou mais componentes relacionados a uma mesma solução.

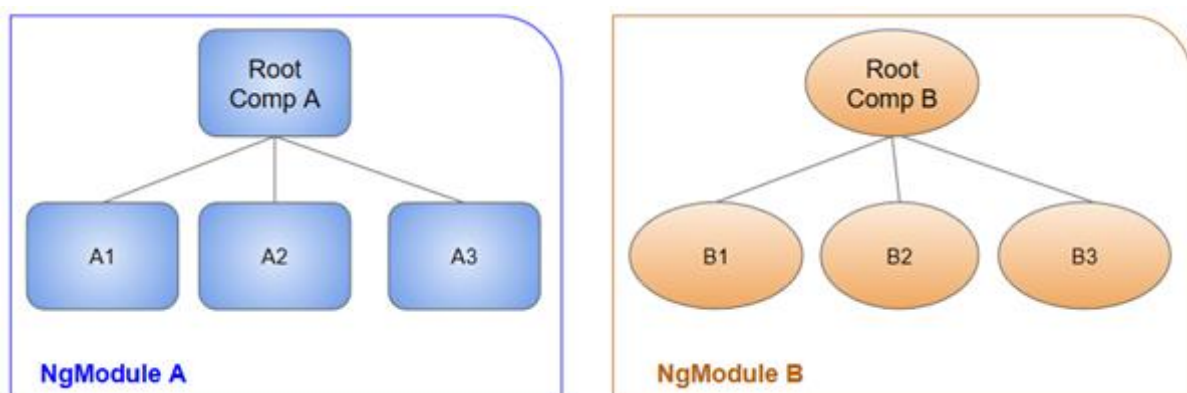


Figura 8 Estrutura de Módulo

Conforme ilustrado na Figura 8, um módulo, *Root Comp A*, contém um e somente componente raiz, mas pode interagir com vários componentes. Por sua vez, uma aplicação pode ser composta por vários módulos.

Um componente pode usar tanto seus *templates*, *views*, quanto de outros componentes, vide *Figura 9*.

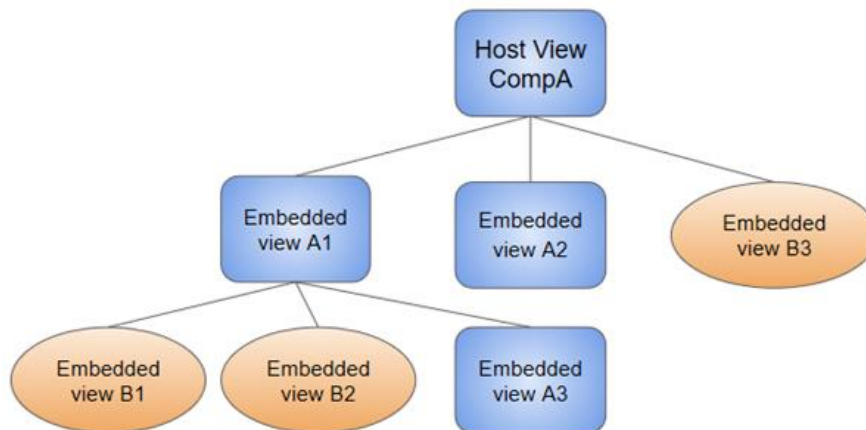


Figura 9 Esquema de uso de template

5.1.2 Componente (*Component*)

Controlar parte da tela, *view*. Cada componente controla apenas eventos, renderização e demais comportamentos de elementos relacionados a sua própria *view*. Um componente poderá se comunicar com outros por meio de mensagens.

A estrutura básica de um componente é composta, mas não limitada a: *Template*, *Service*, Funções.

5.1.2.1 HTML (*Template*)

Representação visual do conteúdo manipulado pelo componente. O *template* deverá ter arquivo próprio para agrupar todo HTML necessário à sua representação.

5.1.2.2 Service

Um serviço pode estar vinculado a um componente, nesse caso, deverá ser vinculado a própria estrutura do componente ou servir a vários componentes distintos, quando assim, deverá ser estruturado de forma global. Ter classe própria e objetivo específico.

5.1.2.3 Funções

Agrupar código que não se enquadre em uma *service*, mas se faz necessário a um componente.

5.2 Front-end

Esta seção define a lista de clientes, navegadores, que a camada de Front-end deverá suportar.

A *Tabela 4* lista os navegadores que deverão ser suportados pelas aplicações Front-end CAPES. Isso implica que todas as *tags HTML*, *CSS* e funcionalidades Javascript empregadas na construção de telas e comportamentos/eventos deverão ter *status* de *Supported* por seus respectivos fabricantes.

Navegador Desktop	Versão mínima
Google Chrome	73
Mozilla Firefox	66
Apple Safari	12
Microsoft Edge	12

Tabela 4 Lista de clientes suportados pelas aplicações CAPES

A verificação do suporte de *tag* HTML, CSS e Javascript pode ser realizada em: <https://caniuse.com>.

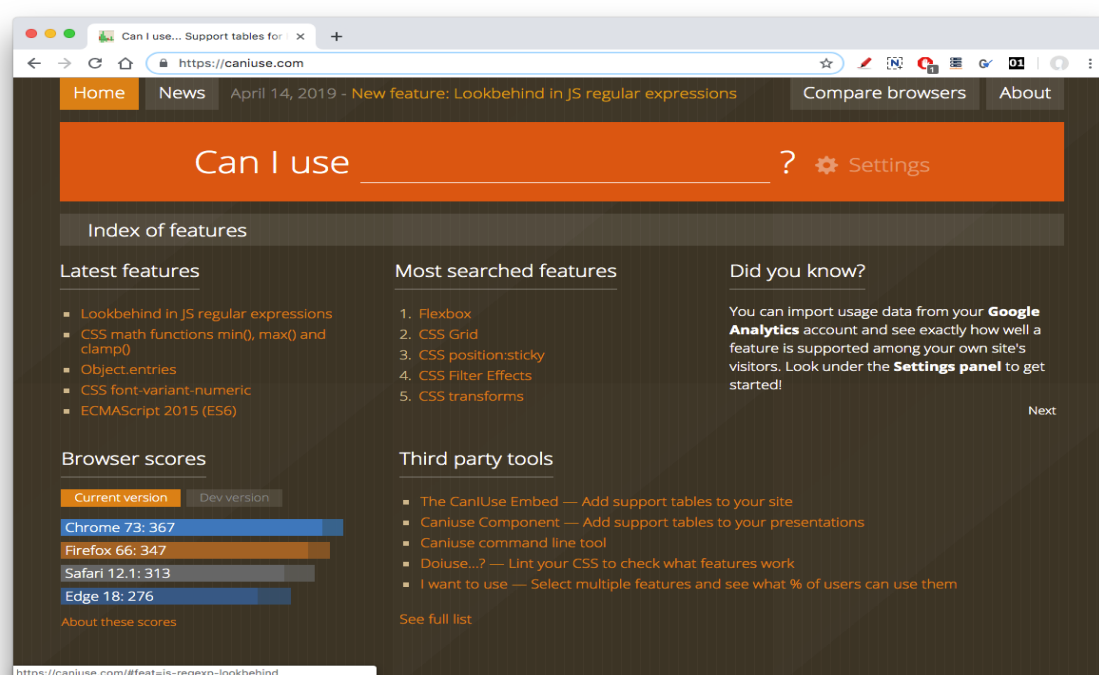


Figura 10 URL caniuse

5.3 Componente

A CAPES já possui alguns componentes de estrutura básica para auxiliar o desenvolvimento da interface do sistema.

Os componentes desenvolvidos apenas complementam a arquitetura de componentes já existente nas aplicações, de forma a acelerar o desenvolvimento, entretanto podem ser substituídos caso seja necessário.

Os componentes desenvolvidos auxiliam o suporte à validação de campos de tela, processamento de notificações, logging, autenticação e acesso a serviços de backend via rest.

6. Requisitos de Qualidade

O Índice de Qualidade (IQ) será obtido a partir de indicadores de qualidade do código-fonte do software aferidos com o apoio da ferramenta SonarQube. Alguns pontos que serão mais cuidadosamente analisados com base nos relatórios do sonar são: cobertura de testes, complexidade ciclomática, acoplamento e coesão.

O acesso ao SonarQube é livre para todos os desenvolvedores e deve ser incentivado dentro das equipes como meio de adequação e qualidade de código.

Além dos itens analisados pelo SonarQube, o código deverá ser submetido a testes de carga (estresse) pela ferramenta Apache JMeter a qual demanda script JMeter capaz de produzir o referido teste.

7. Definições, acrônimos e abreviações

- Backend – Código executado no lado do servidor
- *Frontend* – *Código executado no lado cliente*
- Jenkins – <http://jenkins.capes.gov.br>
- Nexus – <http://nexus.capes.gov.br>
- Sonar – <http://sonar.capes.gov.br>
- GITLab – <http://git.capes.gov.br>
- Wiki – <http://wiki.capes.gov.br>
- Angular – <https://angular.io>
- React – <https://reactjs.org>
- Vuejs – <https://vuejs.org>
- JSON – <https://json.org>

8. Referências

- https://wiki.capes.gov.br/index.php/DTI:Politica_de_Geracao_de_Builds_para_Deploy
- https://wiki.capes.gov.br/index.php/DTI:Visao_geral_tecnologias_integradas
- Crie aplicações com Angular: O novo framework do Google, Thiago Guedes, Casa do código
- Guia Front-End: O caminho das pedras para ser um dev Front-end, Diego Eis, Casa do código
- Building Isomorphic JavaScript Apps: From Concept to Implementation to Real-World Solutions, Maxime Najim, Jason Strimpel, O'Reilly
- Angular, disponível em: <https://angular.io>, acessado em: 08/04/2019
- Angular + Jasmine, disponível em: <https://medium.com/angularbr/angular-5-criando-testes-com-jasmine-175170612ed8>, acessado em: 15/04/2019
- SPA, disponível em: <https://blog.pshrmn.com/entry/how-single-page-applications-work>, acessado em: 10/04/2019
- REST, disponível em: <https://restfulapi.net>, acessado em: 10/04/2019
- REST - Restrições, disponível em: 10/03/2019 <https://www.restapitutorial.com/resources.html>, acessado em: 15/04/2019
- REST - Princípios e boas práticas, disponível em: <https://blog.caelum.com.br/rest-principios-e-boas-praticas/>, acessado em: 10/04/2019
- MCV, disponível em: <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>, acessado em: 15/04/2019
- JSON, disponível em: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>, acessado em: 08/04/2019
- IoC, disponível em: https://pt.wikipedia.org/wiki/Inversão_de_controle, acessado em: 08/04/2019
- CDN, disponível em: <https://www.keycdn.com/what-is-a-cdn>, acessado em: 09/04/2019
- HTML5, disponível em: <https://html5forwebdesigners.com/history/index.html>, acessado em: 24/04/2019
- HTTP Status, disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>, acessado em: 19/04/2019
- HTTP Methods, disponível em: <https://www.restapitutorial.com/lessons/httpmethods.html>, acessado em: 19/04/2019
- MVC/MVP/MVVM, disponível em: <https://medium.com/@FilipeFNunes/android-mvc-x-mvp-x-mvvm-qual-pattern-utilizar-parte-1-3defc5c89afd>, acessado em: 22/04/2019
- SOFEA, disponível em: dzone.com/articles/service-oriented-front-end-architecture-sofea, acesso em: 08/04/2019

9. Anexo A – Tecnologias

NOME	VERSÃO	DESCRIÇÃO
PHP	7.3+	Interpretador PHP

10. Anexo B – Componentes

NOME	VERSÃO	DESCRIÇÃO
@capes/ngx	3.1.2	Este projeto agrega todas as dependências necessárias para a montagem de projetos angulares.
@capes/ngx-validation	3.1.1	Contém regras de validação para uso em formulários reativos.
@capes/ngx-auth	3.1.1	Contém os códigos necessários para o processamento de autenticação e autorização.
@capes/ngx-base	3.1.1	Contém as classes de base (template method) padronizando a construção de interfaces em angular.
@capes/ngx-rest	3.1.1	Contém as classes de base para a construção de serviços de comunicação via rest com o backend.
@capes/ngx-ui-forms	3.1.1	Este pacote tem os componentes usados para processamento de formulários.
@capes/ngx-forms	3.1.1	Componente que cria controles para formulários reativos.
@capes/ngx-notification	3.1.1	Este pacote permite o uso de notificações em aplicações, trata-se de um componente do pacote @capes/ngx e deve

NOME	VERSÃO	DESCRIÇÃO
		ser instalado pelo @capes/ngx.
@capes/ngx-logging	3.1.1	Contém os componentes para gerenciamento de logging.
@capes/ngx-ui-base	3.1.1	Contém classes base para construção de componentes de interface.
@capes/ngx-ui-mask	3.1.1	Este pacote adiciona suporte de mascaras em aplicações angulares.
@capes/ngx-memory-storage	3.1.1	Memory storage resolve o problema de abertura de abas no browser.
@capes/ngx-utils	3.1.1	Contém utilitários a programação.