

ARQUITETURA DE REFERÊNCIA - GUIA PHP

Versão: 1.1

Histórico de Revisões			
Data	Versão	Descrição	Responsável
07/03/2019	1.0	Criação do Guia Específico de PHP	Harrysson Gilgamesh de Medeiros Nóbrega José Augusto de Jesus Thiago Adelino de Melo Valdecy Lourenço de Araújo Júnior
28/03/2019	1.1	Homologação do Guia Específico de PHP	Alexandre Queiroz de Oliveira Carlos Alberto Rodrigues Santana José Arthur Souza de Macedo José Augusto de Jesus Leonardo Moraes Borges Théo Alves Monteiro Valdecy Lourenço de Araújo Júnior
13/05/2019	1.2	Alteração do quadro de métrica de qualidade no item 4 – Metas e restrições arquiteturas.	Alexandre Queiroz de Oliveira Carlos Alberto Rodrigues Santana José Arthur Souza de Macedo José Augusto de Jesus Théo Alves Monteiro Davi Souza Rafael Jesus Nirian Martins S. dos Santos

SUMÁRIO

1.	INTRODUÇÃO.....	5
2.	PROPÓSITO.....	5
3.	ESCOPO.....	5
4.	METAS E RESTRIÇÕES ARQUITETURAIS.....	5
4.1	CAMADA.....	6
4.1.1	CAMADA DE RECURSOS.....	7
4.1.2	CAMADA DE NEGÓCIO.....	7
4.1.3	CAMADA DE ACESSO A DADOS.....	7
4.2	FRONTEND.....	8
4.3	BACKEND.....	8
4.4	PADRÕES.....	10
4.4.1	TECNOLOGIA.....	10
4.4.2	COMPONENTE.....	11
4.4.3	NOMENCLATURA.....	11
5.	VISÃO LÓGICA.....	11
6.	VISÃO DE PROCESSOS.....	11
7.	VISÃO DE IMPLANTAÇÃO.....	11
8.	VISÃO DE IMPLEMENTAÇÃO.....	11
8.1	MICROSSERVIÇOS.....	12
8.2	DECOMPENDO APLICAÇÕES EM SERVIÇOS.....	12
8.3	IMPLANTAR MICROSSERVIÇOS DEPENDE DE UM BOM PARTICIONAMENTO.....	14
9.	TAMANHO E DESEMPENHO.....	14
10.	REQUISITOS DE QUALIDADE.....	15
11.	DEFINIÇÕES, ACRÔNIMOS E ABREVIACÕES.....	17
12.	REFERÊNCIAS.....	18

ANEXO A – TECNOLOGIAS	20
ANEXO B – COMPONENTES	21

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 sistemas PHP na CAPES. Detalhar os aspectos técnicos relativos à adoção de tecnologias, ao desenvolvimento e à implantação de sistemas. Capturar e formalizar as principais decisões tomadas com relação à arquitetura de sistemas que tenham o PHP como base. 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.

3. Escopo

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

4. Metas e Restrições Arquiteturais

Indicadores de qualidade que cada sistema deverá apresentar ao ser submetido à avaliação. Cada indicador representa 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

Grupo	Indicador	Tipo Meta	Meta
	Sucesso em testes unitários (%)	%	= 100%
	Cobertura (camada de negócio)	%	>= 70%

Em alinhamento com o Guia Geral, a arquitetura de aplicações PHP é projetada para o desenvolvimento em microsserviços e é agrupada em dois segmentos: API e DevOps.

- API é o agrupamento dos pontos de consumo da aplicação que podem ser invocados por outros sistemas ou interfaces de usuários especializadas. Constitui a fronteira entre o *backend* e *frontend*.
- DevOps é o conjunto de scripts e artefatos necessários à automação do processo build/deploy da aplicação usando pelo OpenShift.

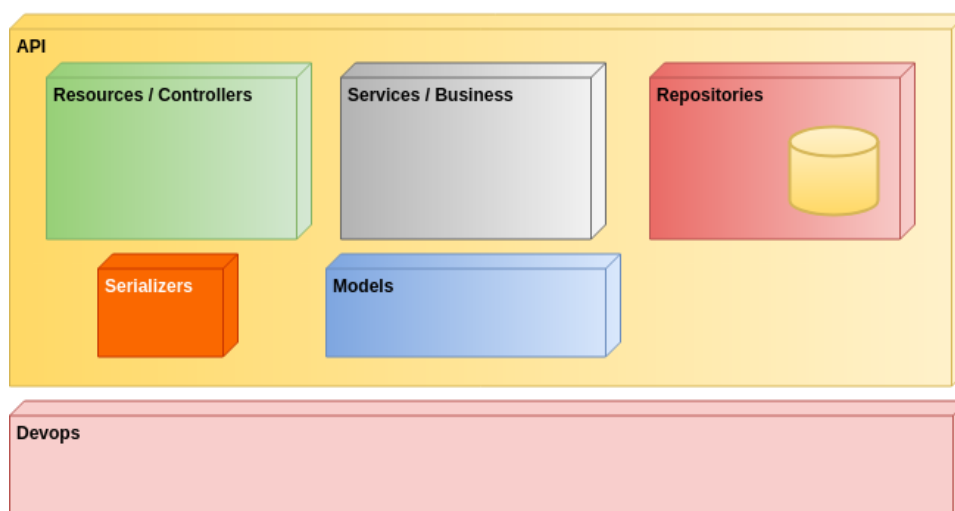


Figura 1 Arquitetura de aplicação PHP

4.1 Camada

Conforme mostrado na *Figura 1 Arquitetura de aplicação PHP*, o primeiro seguimento da arquitetura, API, agrupa as camadas que compõem às aplicações. Nesse contexto, cada camada exerce papel específico e bem definido no plano de codificação. Suas responsabilidades são agrupadas em 3 grupos:

- **Resources Layer / Controllers / Recursos:** Essa camada é responsável por separar a estrutura de acesso dos dados da camada de apresentação do sistema. Ela centraliza a complexidade requisições e retornos da API Rest. Esta camada adere ao item 3.2.1.1.9 do guia geral.

- **Business Layer / Services / Serviços:** Responsável por centralizar e validar as regras negociais utilizadas para atingir o objetivo da aplicação. Esta camada também tem como objetivo centralizar o acesso a serviços externos, sendo estes em sua maioria serviços web Rest e SOAP. Como exemplo de serviço externo consumido está o Cadastro de Pessoas, o Serviço de Documentos e o Serviço de Pessoas da Receita. Esta camada adere aos itens 3.2.1.1.5 e 3.2.1.2.3 do guia geral.
- **Repositories Layer / Camada de banco de dados:** Representa o SGBD ou local de persistência dos dados. Podendo ser um banco de dados relacional (Oracle, Postgresql) ou não relacional (MEMCached, ElasticSearch). Esta camada adere aos itens 3.2.1.1.4 e 3.2.1.2.1 do guia geral.

4.1.1 Camada de recursos

O conjunto de rotinas e o de padrões estabelecidos e documentados da API devem ser implementados nessa camada. Ela é o ponto de partida das funcionalidades do microserviço.

Deve ser utilizado um dos frameworks homologados pela CAPES, vide Anexo A– Tecnologias, para descrever a API e expor sua documentação utilizando o Swagger.

Para cada *namespace* da API, dever-se-á criar um *Resource*, que descreve e implementa o acesso à camada de negócios, processa e serializa dados de retorno de um objeto *HttpResponse* contendo o conteúdo do modelo serializado solicitado ou gera uma exceção como *Http404*.

4.1.2 Camada de negócio

Todo acesso à camada de Banco de dados deve ser feito a partir da camada de negócio, nela também são verificadas e validadas as regras negociais, podendo emitir exceções se for o caso. Devem ser consumidos todos os tipos de serviços externos, independente se são serviços web usando REST ou SOAP. Essa camada tem como objetivo abstrair o consumo desses recursos, deixando visível para as outras camadas apenas os parâmetros de requisição e o retorno esperado. Por fim, controla as transações quando as operações envolverem manipulação de dados.

4.1.3 Camada de acesso a dados

Implementação do modelo de dados no PHP definido em conjunto com a equipe de banco de dados e suas boas práticas. Os dados presentes nessa camada devem ser utilizados pela camada de acesso a dados.

Na CAPES, os sistemas são isolados, cada um tem um esquema de banco de dados diferente, um sistema não deve ter acesso a dados de outro sistema via banco de dados. A maneira que estes dados devem ser compartilhados estão descritos no item "Camada de negócio" deste documento.

Todo acesso ao banco de dados no esquema do sistema é feito através da camada de acesso a dados. As consultas DML de recuperação, inclusão, remoção e modificação de informações em bancos de dados devem ser feitas através de técnica de mapeamento objeto relacional que permita fazer uma relação dos objetos com os dados os quais representam (*Object Relational Mapper*).

As aplicações desenvolvidas na CAPES utilizam como mecanismo de persistência SGDBs relacionais. As aplicações PHP fazem uso do framework Doctrine para acessar os dados dos bancos.

4.2 Frontend

Não se aplica.

4.3 Backend

Sem prejuízo à definição do Guia Geral 3.1.3. A seguir, é definido como uma aplicação PHP será executada em ambiente CAPES.

No modelo tradicional de execução, para cada requisição ao servidor web, por exemplo Apache, é criada uma *thread* e, nela, uma instância do PHP. Esse modelo se mostra demasiadamente custoso para o processo.



Figura 2 Modelo de execução tradicional do PHP

Com o modelo da de execução tradicional do PHP, Figura 2 Modelo de execução tradicional do PHP, o atendimento a grande volume de requisição tende a exaurir os recursos do servidor rapidamente ou provocar lentidão nesse processo.

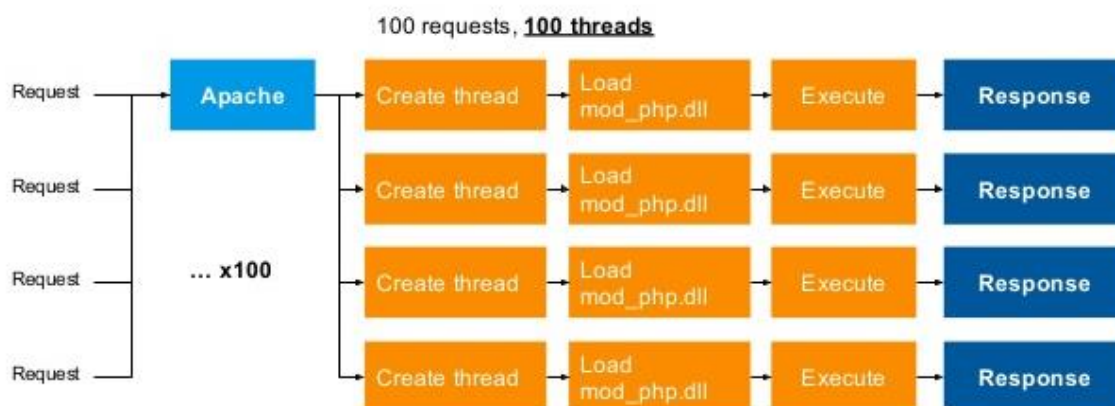


Figura 3 Alto número de threads

A Figura 3 Alto número de threads mostra o alto custo de recursos empregados para suprir a demanda de requisições conforme o número delas cresce.

Porém, desde sua versão 5.3.3, o PHP oferece suporte ao *FastCGI Process Manager* (FPM ou PHP-FPM). Neste modelo, o servidor web tem papel menos protagonista e atuar mais como um serviço de Proxy.

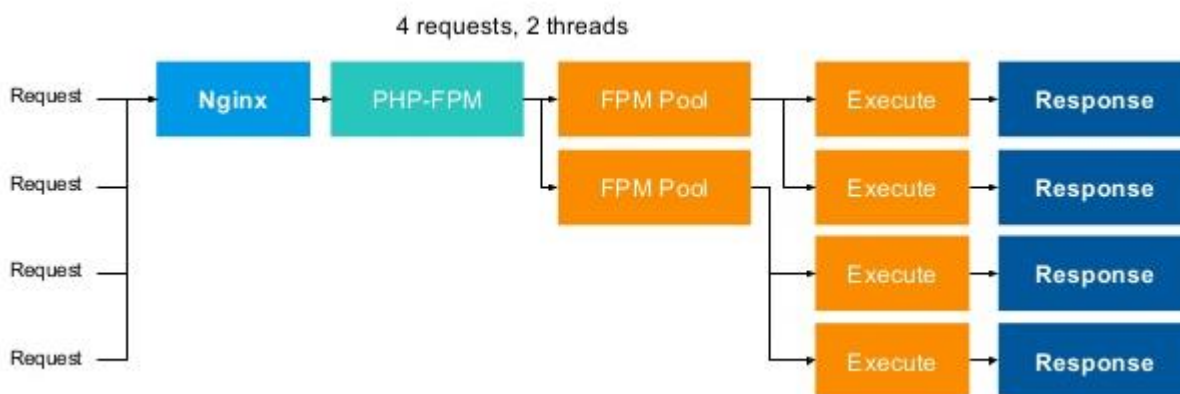


Figura 4 Modelo de execução PHP-FPM

Com o PHP-FPM, o número de *threads* do servidor web se descola do número de requisições e isso torna o processo mais célere e consome menos recursos do servidor. A Figura 4 Modelo de execução PHP-FPM evidência que há uma instância de servidor web ligada a um serviço PHP-FPM, que, por sua vez, possui um *pool* de *threads* prontas para executar as requisições.

A CAPES adotará o modelo PHP-FPM para execução de aplicações codificadas em PHP.

4.4 Padrões

Definições adotadas pela CAPES com base em suas necessidades para criação, manutenção e evolução de seus sistemas. A CAPES adere aos padrões especificados pela comunidade PHP: *PSR (PHP Standards Recommendations)*. Assim, como regra, todo código PHP deverá atender aos requisitos de seu ou seus *PSRs*. A listagem completa e atualizada dos *PSR* está disponível em: www.php-fig.org.

Para o especificado acima, dever-se-á considerar apenas as *PSR* que tenha *status* 'ACCEPTED'.

ACCEPTED				
NUM	TITLE	EDITOR	COORDINATOR	SPONSOR
1	Basic Coding Standard	Paul M. Jones	N/A	N/A
2	Coding Style Guide	Paul M. Jones	N/A	N/A
3	Logger Interface	Jordi Boggiano	N/A	N/A
4	Autoloading Standard	Paul M. Jones	Phil Sturgeon	Larry Garfield
6	Caching Interface	Larry Garfield	Paul Dragoonis	Robert Hafner
7	HTTP Message Interface	Matthew Weier O'Phinney	Beau Simensen	Paul M. Jones
11	Container Interface	Matthieu Napoli, David Négrier	Matthew Weier O'Phinney	Korvin Szanto
13	Hypermedia Links	Larry Garfield	Matthew Weier O'Phinney	Marc Alexander
14	Event Dispatcher	Larry Garfield	N/A	Cees-Jan Kiewiet
15	HTTP Handlers	Woody Gilk	N/A	Matthew Weier O'Phinney
16	Simple Cache	Paul Dragoonis	Jordi Boggiano	Fabien Potencier
17	HTTP Factories	Woody Gilk	N/A	Matthew Weier O'Phinney
18	HTTP Client	Tobias Nyholm	N/A	Sara Golemon

Figura 5 Quadro de *PSRs* com status *ACCEPTED*

4.4.1 Tecnologia

O Anexo A - Tecnologias elenca as tecnologias adotadas pela CAPES, de forma não exaustiva, para a pilha de desenvolvimento PHP. A inclusão ou remoção de itens nele, deverá ser definido pela CAPES.

4.4.2 Componente

O Anexo B - Componentes elenca os componentes adotados pela CAPES para a pilha de desenvolvimento PHP. A inclusão ou remoção de itens nele, deverá ser definido pela CAPES.

4.4.3 Nomenclatura

A CAPES adotará o padrão PSR-2 para o desenvolvimento em PHP.

5. Visão lógica

Vide correlação no item 4.1.

6. Visão de processos

Não se aplica.

7. Visão de implantação

Atualmente, os ambientes não produtivos e produtivos da Capes estão disponibilizados via Openshift. De modo que é possível baixar uma imagem de um contêiner Docker homologada pela equipe de infraestrutura e iniciar o desenvolvimento diretamente no container sem a necessidade de que seja provido um ambiente específico para desenvolvimento ou servidor pré-instalado.

Para a geração do pacote de *deploy* da aplicação, a Pipeline do GitLab se integra com o Jenkins. No GitLab, é feita a criação de uma TAG que represente o código usado na versão do sistema que será manipulado. A cada execução dessa tarefa, faz-se uma análise estática de código através do SonarQube. Por fim, o pacote é gerado e disponibilizado no contêiner que seguirá todos os passos de aceitação para ser disponibilizado em ambiente produtivo.

Na Wiki, existe uma página que detalha as configurações que devem ser feitas pelo desenvolvedor na sua máquina e no projeto. A página está disponível em [https://wiki.capes.gov.br/index.php/DTI:Politica de Geracao de Builds para Deploy](https://wiki.capes.gov.br/index.php/DTI:Politica_de_Geracao_de_Builds_para_Deploy)

Cada projeto deverá ter um pacote chamado **devops** em seu repositório, que é mantido pela equipe de arquitetura e pelo líder técnico da equipe. Neste pacote deve conter os arquivos de CI e CD da aplicação.

8. Visão de implementação

PHP é uma linguagem de programação poderosa e de fácil aprendizado. Ela possui estruturas de dados de alto nível e adota uma abordagem simples e efetiva para a programação orientada a objetos. Sua sintaxe elegante e tipagem dinâmica, em adição à sua natureza interpretada, as tornam

ideal para *scripting* e para o desenvolvimento rápido de aplicações em diversas áreas e em várias plataformas.

A extensa biblioteca padrão do PHP está disponível na forma de código fonte aberto a partir do PECL (*PHP Extension Community Library*) ou por componentes prontos e disponibilizados por seu gerenciador de pacotes mais popular – Composer.

O interpretador de PHP é facilmente extensível incorporando novas funções e tipos de dados implementados em C ou C++ seguindo o modelo de extensão disponível no site oficial do próprio PHP.

Na comunidade PHP, existem uma grande quantidade de frameworks disponíveis para serem utilizadas, porém na CAPES, foram adotados os frameworks que constam no **Anexo – Tecnologias**.

8.1 Microsserviços

Tradicionalmente, softwares são construídos como uma estrutura fechada, com começo, meio e fim. Ou melhor, *backend* e *frontend*.

Essa filosofia acaba gerando peças muito grandes (Arquitetura Monolítica). Normalmente, focadas em resolver grandes necessidades e problemas organizacionais. Como exemplo, sendo necessário usar somente as funções que fazem gráficos do Excel para alguma outra aplicação, terá que abrir o Excel e carregar outras funcionalidades desnecessárias naquele momento.

A Arquitetura de Microsserviços, ao contrário das anteriores de Arquitetura Monolítica ou até SOA, destaca-se por explorar a ideia de granularidade, o que facilita a execução do próprio serviço e a adaptação às mudanças.

A ideia é dividir um determinado sistema em serviços acionáveis e modulares de modo que a união de pequenas partes realize um trabalho maior. Assim, os microsserviços permitem a integração entre vários serviços e a inserção de vários componentes no sistema.

8.2 Decompondo aplicações em serviços

A forma mais usual para escalar uma aplicação é executando várias cópias idênticas de aplicação (por meio de um balanceador de cargas) realizando um processo chamado Decomposição.

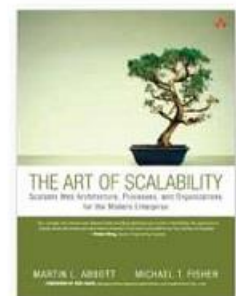
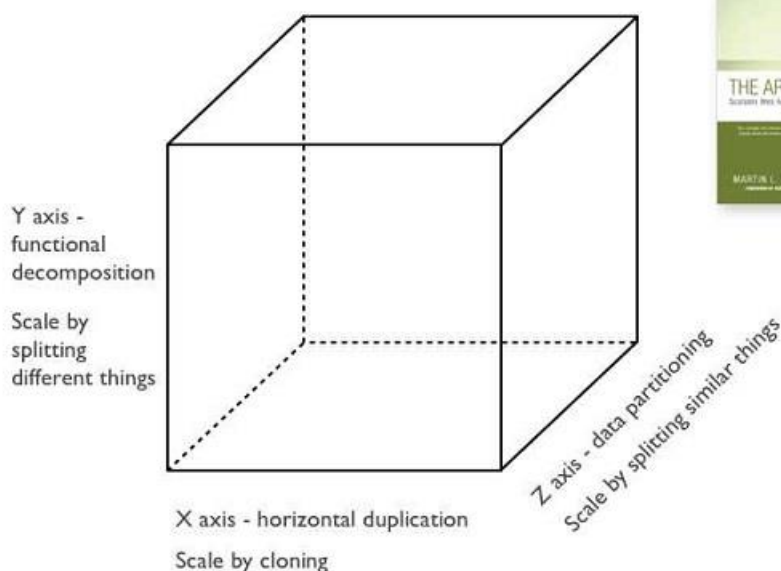
A finalidade da decomposição é resolver problemas comuns na arquitetura monolítica, implicando em que alguns serviços poderão ser muito pequenos enquanto outros serão significativamente maiores.

No conceito de três dimensões, é possível entender como os Microsserviços são úteis no particionamento e escalabilidade de uma aplicação monolítica. Temos três eixos de escalabilidade:

- X: referente à escalabilidade horizontal, para ampliar a capacidade e disponibilidade da aplicação (cada servidor executa uma cópia idêntica do código);
- Z: semelhante à do eixo X, mas requer a presença de um componente que se responsabilize pelo roteamento das requisições ao servidor adequado;
- Y: é a terceira dimensão da escalabilidade, a horizontal, denominada decomposição funcional e é responsável por dividir a aplicação em uma série de serviços. A cada serviço corresponde um conjunto de funções (gerenciamento de pedidos, gerenciamento de clientes e assim por diante).

Lembrando que, enquanto o eixo Z divide elementos semelhantes, o eixo Y divide elementos distintos.

3 dimensions to scaling



8.3 Implantar Microsserviços depende de um bom particionamento

A divisão, ou particionamento, do sistema deve ser bem elaborada, segue as abordagens que adotamos para implantar a Arquitetura de microsserviços:

- **Por verbos/casos de uso:** o caso *Checkout*, por exemplo, em que um serviço de conclusão de pedidos, ou *Checkout UI service*, implementa a interface com as pessoas que usam o sistema.
- **Por sinônimos/recursos:** como exemplo, considere que, para gerenciamento do catálogo de mercadorias, a empresa pode ter o *Catalog Service*. Nesse caso, o serviço se responsabiliza por todas as atividades que envolvem os recursos/entidades relacionados.

O importante é que cada serviço possua poucas responsabilidades, de acordo com o Princípio de Responsabilidade Única (SRP, *Single Responsibility Principle*), ou seja, um serviço exposto deve ter um único e claro papel dentro da Arquitetura.

Se for o caso em que um serviço possui mais de uma responsabilidade, deve-se aplicar algum particionamento, como citado acima.

As funcionalidades Unix constituem outro exemplo de modelagem de serviços, em que cada funcionalidade realiza somente uma operação definida (podendo, no entanto, ser combinada, por meio de *shell script*, com outras funcionalidades a fim de realizar atividades mais complexas). Ao longo dos anos, esse baixo acoplamento facilitou que diversas variações

9. Tamanho e desempenho

Caso a aplicação tenha processamento em lote ou execuções agendadas, essas operações devem, preferencialmente, ser realizadas em servidores separados dos servidores destinados a atender a requisições dos usuários. Devido aos custos de processamento destes recursos, a resposta do usuário pode ser onerada caso estejam no mesmo servidor.

As requisições devem ser atendidas o mais rápido possível. Processamentos de pedidos de usuários que gerem demora perceptível na *request* deve ter o seu processamento efetuado em *background* e o usuário deve ser notificado da conclusão da tarefa.

Preferir a separação do *front-end* e do *back-end*, visando facilitar a escalabilidade de recursos e o desacoplamento entre tecnologias com fins diferentes.

10. 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.

Com a finalidade de garantir uma boa qualidade do código desenvolvido, os desenvolvedores devem usar o BDD (Design/Desenvolvimento guiado por comportamento), que em resumo é uma técnica voltada para o comportamento da aplicação e é comumente usada para testes.

Como forma de auxiliar no desenvolvimento dos códigos, podem ser utilizadas, bibliotecas que facilitem a atividade de teste, entre elas:

- **PHPUnit** - framework usado para escrever testes unitários de forma que permita a sua execução automática.
- **PHPUnit** – framework para executar testes de integração, auxiliando em testes que precisem de um banco ou recursos de container.
- **PHPUnit** – framework para executar validação de métricas CRAP (*Change Risk Analysis and Predictions*)
- **PHPCS (PHP CodeSniffer)** – ferramenta para indicar violação às regras do padrão de estilo de codificação do código PSR-2.
- **PHPMD (PHP Mess Detector)** – ferramenta para indicar possíveis bugs ou uso indevido da linguagem.
- **PHPMND (PHP Magic Number Detector)** – ferramenta para detectar números mágicos no código.

- **PHPStan (PHP Static Analysis Tools)** – ferramenta para avaliar o código estaticamente revelando erros que são mostrados em tempo de compilação (execução).
- **PHPCPD (PHP Copy Past Detector)** – ferramenta para detectar bloco de código duplicado.
- **CHURN-PHP** – ferramenta para detectar classes que precisam ser refatoradas com base nas métricas de *cyclomatic complexity*.
- **Selenium** – ferramenta usada para mapear a navegação do usuário, usada para escrever os testes de aceitação.
- **Apache JMeter** – ferramenta para execução de carga estresse no código.

11. 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>
- Openshift - <http://openshift.capes.gov.br>
- Lumen - <https://lumen.laravel.com>
- Slim - <http://www.slimframework.com>
- Doctrine - <https://www.doctrine-project.org>
- Swagger - <https://swagger.io>
- Angular - <https://angular.io>
- React - <https://reactjs.org>
- Vuejs - <https://vuejs.org>

12. Referências

- [https://wiki.capes.gov.br/index.php/DTI:Politica de Geracao de Builds para Deploy](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](https://wiki.capes.gov.br/index.php/DTI:Visao_geral_tecnologias_integradas)
- The Art of Computer Programming: D. E. Knuth, Addison-Wesley (volumes 1--3, 4A), 1998.
- Composer, disponível em: getcomposer.org, acessado em: 11/03/2019
- PHP, disponível em: php.net, acessado em 11/03/19
- PHP-FPM, disponível em: http://php.net/manual/pt_BR/install.fpm.php, acessado em: 22/03/2019
- PSR, disponível em: www.php-fig.org/psr/, acessado 20/03/2019
- PHPUnit, disponível em: phpunit.de | phpunit.readthedocs.io, acessado em 11/03/19
- PHPCS, disponível em: github.com/squizlabs/PHP_CodeSniffer/wiki, acessado em: 20/03/2019
- PHPMD, disponível em: phpmd.org, acessado em: 20/03/2019
- PHPMND, disponível em: github.com/povils/phpmnd, acessado em: 20/03/2019
- PHPStan, disponível em: github.com/phpstan/phpstan, acessado em: 20/03/2019
- PHPCPD, disponível em: github.com/sebastianbergmann/phpcpd, acessado em: 20/03/2019
- CHRUN-PHP, disponível em: github.com/bmitch/churn-php, acessado em: 20/03/2019
- Apache JMeter, disponível em: jmeter.apache.org, acessado em: 20/03/2019
- Software Metrics for Code Coverage, disponível em: phpunit.de/manual/6.5/en/code-coverage-analysis.html, acessado em: 20/03/2019
- Cyclomatic complexity, disponível em: en.wikipedia.org/wiki/Cyclomatic_complexity, acessado em: 20/03/2019

- CRAP Metric Is a Thing And It Tells You About Risk in Your Code, disponível em: blog.ndepend.com/crap-metric-thing-tells-risk-code/, acessado em: 20/03/2019
- How to Read and Improve the C.R.A.P Index of your code, disponível em: opnsrce.github.io/how-to-read-and-improve-the-c-r-a-p-index-of-your-code, acessado em: 20/03/2019
- How to Read / Improve C.R.A.P Index Calculated by PHP, disponível em: stackoverflow.com/questions/4731774/how-to-read-improve-c-r-a-p-index-calculated-by-php, acessado em: 20/03/2019

Anexo A – Tecnologias

NOME	VERSÃO	DESCRIÇÃO
PHP	7.3+	Interpretador PHP
Composer	1.8+	Gerenciador de dependências
Lumen	5.8+	Micro framework baseado em Laravel
Slim	3+	Micro Framework PHP
Symfony	4.3+	Framework PHP
Doctrine (ORM)	2.6+	ORM PHP
Swagger-PHP	3+	Gerador de API Swagge http://zircote.com/swagger-php/
Codeception	2.1+	Framework para testes
PHPUnit	8+	Framework para testes
Monolog	1.24+	Componente de manipulação de Log https://github.com/Seldaek/monolog
JWT	1.3+	Componente para manipulação de Token https://github.com/web-token/jwt-framework

Anexo B – Componentes

NOME	VERSÃO	DESCRIÇÃO
-	-	-