**MINISTÉRIO DE MINAS E ENERGIA**
**Secretaria Executiva**
**Subsecretaria de Tecnologia e Inovação**
**Coordenação Geral de Inovação e Transformação Digital**

# Padrão de nomenclatura e codificação

## Versão 1

# Sumário

**1. INTRODUÇÃO ..... 3**
**2. PRINCÍPIOS DE NOMENCLATURA ..... 3**
2.1 USE NOME PASSIVÉIS DE BUSCA ..... 3
2.1 2.2 EXPRESSE INTENÇÕES E FACILITE A COMUNICAÇÃO ..... 4
**3. DOCUMENTAÇÃ0 DOS OBJETOS ..... 5**
3.1 NOME DE CLASSES ..... 5
NOME DE CLASSES - JAVA ..... 6
NOME DE INTERFACE ..... 6
NOME DE INTERFACE - JAVA ..... 6
3.2 NOME DE MÉTODOS E ATRIBUTOS ..... 6
NOME DE MÉTODOS E ATRIBUTOS - JAVA ..... 7
3.3 CONSTANTES ..... 7
3.4 COMENTÁRIOS DE CÓDIGO ..... 7
**4. REST E SEUS PADROES ..... 8**
**5. URL ..... 8**
**6. URI ..... 8**
6.1 PADRÕES URI ..... 8
6.2 FORMATO DO URI ..... 8
6.3 AUTORIDADE DE PROJETO URI ..... 9
6.4 MODELAGEM DE CÓDIGO ..... 10
6.5 ARQUÉTIPOS DE RECURSOS ..... 10
6.6 DESIGNER DOS CAMINHOS URI ..... 10
6.7 DESIGNER DE CONSULTA ..... 11
6.8 MÉTODOS DE REQUISIÇÃ0 ..... 12
6.9 CÓDIGO DOS STATUS DE RESPOSTA ..... 12
6.10 API'S REST ..... 13
**7. CONCLUSÃO ..... 13**
**6. REFERÊNCIAS ..... 13**

# 1. INTRODUÇÃO

O maior esforço do desenvolvedor de software é dispensado às atividades de manutenção, desta forma quanto mais fácil for o entendimento do código do sistema, mais produtiva será a equipe de desenvolvimento. Frequentemente as pessoas que escrevem o código não são as mesmas que o mantém e, quando são, geram uma dependência com o código desenvolvido que dificilmente é dissolvida. Um padrão de codificação visa minimizar esses problemas, pois estabelece regras definindo como o código deve ser escrito para favorecer a impessoalidade do artefato, aparentando ser escrito sempre pela mesma pessoa.

Este documento tem como objetivo definir um padrão de codificação que, quando usado, garante um melhor entendimento por qualquer pessoa que siga o mesmo.

Use o bom senso. Quando não houver nenhuma regra ou diretriz disponível, quando a regra não puder ser aplicada por motivos óbvios ou quando todo o resto falhar, use o bom senso e verifique os princípios fundamentais. Essa regra se sobrepõe a todas as outras. Ter bom senso é obrigatório.

# 2. PRINCÍPIOS DE NOMENCLATURA

Os padrões de codificação para qualquer linguagem são de extrema importância, porque garantem maior consistência de seu código e do código de seus companheiros de equipe. Maior consistência gera códigos mais fáceis de serem entendidos, desenvolvidos e mantidos. Dessa forma, é reduzido o custo total dos aplicativos criados.

Lembre-se de que seu código continuará a existir por muito tempo ainda. Uma meta importante durante o desenvolvimento é garantir que se possa passar o trabalho para outro profissional, a fim de que ele possa continuar a manter e melhorar o trabalho sem precisar investir um esforço desnecessário para entender o código que foi criado. Códigos de difícil compreensão correm o risco de serem descartados e reescritos.

Segue nos próximos tópicos o detalhamento de como fazer um excelente código e de fácil entendimento.

## 2.1 USE NOMES PASSÍVEIS DE BUSCA

Sempre utilize nomes significativos, descrevendo o que a variável representa, nomes que não expressam nenhum sentido é difícil de localização textual e negocial. Quando temos um código fonte que possui várias variáveis com nomes não significativos, como exemplo variável apenas de uma letra, quando um desenvolvedor for executar uma

manutenção no código fonte e que não tenha conhecimento negocial do sistema, o que podemos esperar? Apenas dúvidas, onerando o tempo de trabalho e solicitando as pessoas que conhecem da regra de negócio executar a demanda. Isso nos mostra que além de estar com o código ruim, a organização está a mercê do desenvolvedor, sem ele outra pessoa não consegue dar prosseguimento a demanda.

Variáveis como "a1", não representa nada negocialmente. Para um desenvolvedor que conhece o código pode representar muita coisa, mas para outro desenvolvedor não quer dizer nada, pois o código representa uma tarefa difícil de compreensão e totalmente descartável no primeiro impacto e é onde gera um retrabalho desnecessário, para descartar o código e fazer algo melhor e limpo.

A forma passível de buscar um atributo no código é pela razão dele significar algo, quando temos um nome que expressa sua função, ao efetuar uma busca temos a cognição de encontrar algo usando apenas palavras que expressão algum significado; Exemplo disso é o atributo nomeCarro, quando um desenvolvedor não sabe ou não conhece e tenta efetuar uma busca pela palavra nome, efetivamente encontrara o que está procurando "nomeCarro".

O único fator de usar apenas um caractere como variável é contas matemáticas onde é entendível o valor de x, y e z em fórmulas matemáticas. Exemplo disso a equação de primeiro grau: (ax+b = 0).

O código é a forma de conversar com a máquina e mostrar para seus colegas que você executa um bom código fonte e totalmente entendível.

## 2.2 EXPRESSE INTENÇÕES E FACILITE A COMUNICAÇÃO

Tudo que exige escrita em um código deve ser pensado, onde possa expressar a sua real intenção. Onde o nome deve ser claro, simples e único, o nome escolhido deve ser simples no sentido do contexto do da palavra e que não obscureça o seu significado.

Para facilitar a comunicação os nomes não devem ter segundos significados, ou significado engraçados para a equipe e não devem usar sin6nimos espertinhos em vez da palavra que a esperada. Exemplo disso a terminar execução de um código o nome deste método deve ser terminar() e não darOFora(), pois entender que o método terminar termina algo é mais fácil que darOFora() e é bem mais entendível e comunicável.

Em nomes de conceito técnico, ou usados em designers, as palavras devem conter seu domínio de solução, onde os desenvolvedores entenderam e facilita a escrita explicativa e sua comunicação do código fonte com o resto da equipe. Exemplo: Objeto Carro é um repositório, podemos colocar o nome da interface de CarroRepository, onde a identificação facilita o entendimento inclusive da camada representativa.

A comunicação do seu código fonte é muito importante para a equipe e manutenções futuras, e nomes que mostre sua real intenção é o detalhamento do seu código para todos, pois assim evitamos o retrabalho e a reescrita.

## 3. DOCUMENTAÇÃO DE OBJETOS

Um nome deve ser de fácil entendimento e que realmente revele seu propósito. Pensamos que é fácil de criar um nome que quer mostrar seu prop6sito, mas é complicado e depende do discernimento e entendimento do requisito proposto.

Evite informações erradas no momento de criar um atributo, método, pacote, classe, interface etc., pois ele pode deixar o código fora do padrão e manutenabilidade do sistema.

Usar números sequenciais em nomes é o oposto da seleção de nomes expressivos e significativos. Exemplo: "pessoa" e "pessoal" sac) objetos que fazem a mesma coisa e não justifica a sequência de numeração. Nomes que são curtos, porem expressivos, centralizam a ideia da regra de negócio e o proposito que o código deve executar. Como Brian W. diz "Não insira comentário num código ruim, reescreva-o".

A redundância de código é uma péssima prática para a manutenção e para o projeto, pode atrapalhar a localização do código para efetuar a correção, gerando erros na alteração em regras referentes ao código e levando o desenvolvedor a executar alterações no mesmo código, mas em classes diferentes.

Será apresentado cada objeto e suas particularidades e padrões de cada linguagem. Dessa forma ajudando o desenvolvedor a trabalhar com um código mais limpo e com nomenclaturas legíveis onde cada um pode pegar o código e executar sua leitura.

### 3.1 NOME DE CLASSES

Os nomes das classes não são abreviados, exceto quando são palavras ou nomes extensos, sendo desnecessário colocar todo o nome. Os nomes devem ficar no substantivo e evitar nomes de situação como Processador, Gerador, pois os mesmos não significam nada para outro desenvolvedor, sempre pergunte o nome é Processador, a classe processa o que? Se sabemos responder, já achamos o nome, exemplo processador de threads para cada conexão bancária, podemos dar o nome AtivaConexaoBancaria, neste nome dizemos que a classe ativa a conexão bancária, no caso do cliente ativo.

Os nomes devem ser específicos o suficiente para estarem relacionados ao problema ou a solução.

As classes são definidas a partir da palavra-chave "class", dessa forma estou definindo o início de uma classe.

**NOME DE CLASSES – JAVA**

Nomes de classe Java devem seguir a regra da primeira letra maiúscula, segue um exemplo: Carro, PessoaJuridica, etc.

No Java temos declarações de acesso, onde se usa (public, private, default).

**NOME DE INTERFACE**

Interfaces de Objetos permite a criação de código que especifica quais métodos e variáveis uma classe deve implementar, sem ter que definir como esses métodos serão tratados.

Interfaces são definidas usando a palavra-chave "interface", da mesma maneira que uma classe comum, mas sem nenhum dos métodos ter seu conteúdo definido.

Observação, todos os métodos devem ser public, a natureza da interface é pública, algo privado não pode ser implementado.

Nomes de interface devem ser substantivos, adjetivos e advérbios que designam o conceito ou a responsabilidade que a classe ocupa no sistema. Os nomes devem ser específicos o suficiente para estarem relacionados ao problema ou a solução.

**NOME DE INTERFACE – JAVA**

As regras de nomenclatura de interfaces devem seguir os mesmos princípios das regras para classes. A única diferença é que podemos escolher não apenas substantivo também pode escolher adjetivos e advérbios.

Deve começar com letras maiúsculas o nome da interface e apenas assinaturas de métodos para que no futuro seja implementado, exemplo: Pagamento, Conta, etc.

## 3.2 NOME DE MÉTODOS E ATRIBUTOS

Deve seguir o padrão JavaBeans, o qual segue a regra Camel Case para os atributos e métodos, iniciando com letras minúsculas e as palavras de ligação são colocadas em letra maiúscula. Exemplo: setor, setorLivro, getSetor(), buscaArquivos().

Vale lembrar que método repetido com seu chamador dificulta a manutenção do desenvolvedor, exemplo disso e método addPessoa() na classe FuncaoColaborador, e na classe CargoColaborador ter o mesmo método addPessoa() que executa a mesma coisa mas em classes diferentes, dificulta a manutenção e criamos redundância de código.

De acordo com Robert C. Martin, "use nomes que revelem seu proposito". Dizer que os nomes devem demonstrar seu propósito é fácil. Escolher bons nomes leva tempo, mas economiza mais. Pois todos que lerem seu código (incluindo você mesmo) ficarão agradecidos.

O nome de um atributo, método ou classe deve responder a todos as grandes questões. Ele deve dizer por que existe, o que ele faz e como é usado. Onde identificamos seu verdadeiro proposito. Então é de extrema responsabilidade do desenvolvedor escolher nomes significativos.

Nomes de métodos devem ser verbos, exemplo (enviar, pagar, etc), com a possibilidade de serem compostos com advérbios ou substantivos relativo ao papel dos argumentos (enviarPropostaBanco, etc), métodos que representem acessos ou modificadores devem seguir a nomenclatura camelCase, começando com a primeira palavra minúscula e a pr6xima palavra com a primeira letra maiúscula.

**NOME DE MÉTODOS E ATRIBUTOS – JAVA**

No Java segue o padrão camelCase, mas temos o padrão JavaBeans que necessariamente um atributo que necessite de acesso cria-se um get (acessar) e set (modificador). Exemplos: pagarContaCarteiraBO, getNome(), setNome(), etc.

## 3.3 CONSTANTES

Devem ser escritas em letras maiúsculas, sempre precedidas por public static final para que possamos herdar sem uma instancia da classe. Exemplo: URL_SISTEMA_PROPOSTA.

## 3.4 COMENTÁRIOS DE CÓDIGO

Os comentários constituem apenas uma simplificação para o entendimento do código fonte, mas quem deve realmente expressar o significado é o código fonte e a lógica negocial. E importante que se gaste energia para tornar um código limpo e claro em sua leitura.

## 4. REST E SEUS PADRÕES

REST significa Representational State Transfer em português significa Transferência de Estado Representativo. Como é um estilo de desenvolvimento baseado nos protocolos HTTP 1.1, o envio e recebimento são baseados nos métodos do próprio HTTP 1.1. São: (GET, POST, DELETE, PUT, HEAD, TRACE, OPTIONS e CONNECT).

Para interagir com as URI's, os métodos HTTP são utilizados. A regra de ouro para esta interação e que URI's são substantivos, e métodos HTTP são verbos. Isto quer dizer que os métodos HTTP são os responsáveis por provocar alterações nos recursos identificados pelas URI's. Essas modificações são padronizadas de acordo com:

GET: recupera os dados identificados pelo URI.

POST: cria um recurso.

PUT: altera um recurso existente.

DELETE: deleta um recurso.

Alguns princípios básicos devem ser seguidos para que possamos ter o domínio e trabalhar com a tecnologia REST, são: (uso adequado dos métodos, uso adequado de URL's, usa de códigos corretos para sucesso e falha, uso adequado de cabeçalhos HTTP e interligação entre vários recursos diferentes).

Tudo que é passado pelo endereço é interpretado pelo HTTP, neste caso podemos passar URL ou URI onde temos a diferença entre eles e suas características.

## 5. URL

URL (Universal Resource Locator) é utilizado para fornecer um caminho e ou endereço para identificação e essa identificação é única. Uma URL pode ser considerada um URI de certa forma.

## 6. URI

URI (Universal Resource Identifier) pode ser utilizado para qualquer coisa, como caminho, dar nome a objetos, etc.

### 6.1 PADRÕES URI

Segue os padrões necessários para a construção de um URI, são: (Formato do URI, Autoridade de Projeto URI, Modelagem de Código, Arquétipos de Recursos, Designer dos Caminhos URI e URI de Designer de Consulta).

## 6.2 FORMATO DE URI

As regras apresentadas seguem o formato definido pela RFC3986 como montar de acordo com sua sintaxe genérica, exemplo: URI = esquema"://"autoridade"/"caminho["?"consulta]["#"fragmento].

Regra de separador por barras "/" deve ser usada para identificar hierarquia relacional para indicar um caminho em relação aos recursos liberador hierarquicamente. Exemplo: http://localhost:8080/mme/pessoa/juridica.

Regra de barras no final de um URI é totalmente fora do padrão e não adiciona nenhum valor semântico para a informação. Desta forma transformando o use de barras totalmente fora do padrão e deixando a URI fora do padrão de formatação e acaba mostrando que o código não está limpo o suficiente para identificar um recurso, já que a barra estaria no final. Exemplo: http://localhost:8080/mme/pessoa/juridica/. Fica a pergunta que objeto deve chamar, pois não existe o objeto barra e sim jurídica. Fica confuso e nada semântico.

Regra de hifens “-” deve ser usada para melhorar a legibilidade do objeto e facilitar a interpretação pelos desenvolvedores. Além de evitar outros caracteres especiais ou espaços no objeto para separação da palavra. Exemplo: http://localhost:8080/montadora/automovel/carro/carro-vermelho.

Regra de underscore "_" nao deve ser usada, pois confunde a informação e obscurece a informação como se fosse um sublinhado na palavra. Caso necessite, use a regra acima, referente ao hífen, além de utilizar o padrão irá evitar o obscurecimento da informação.

Regra de URI de caixa baixa é necessária esta utilização, pois a URI é case sensitive e pode causar problemas ao acessar os objetos detalhados no URI. O detalhamento esta descrito na RFC3986, onde mostra as regras case sensitive. Exemplo: http://localhost:8080/carro se colocarmos http://localhost:8080/CARRO, irá executar um erro 404 não encontrado.

Regra de extensão de arquivo não deve se utilizar, pois deve ser configurado no Content-Type como cabeçalho de processamento do corpo do conteúdo.

## 6.3 AUTORIDADE DE PROJETO URI

Agora será tratado o padrão de nomenclatura usado pela API REST.

Regra de nomes de subdomínios consistentes que deve ser usado para API's. Um domínio superior deve identificar seu proprietário e acrescentar seu subdomínio para especificar o caminho da API. Exemplo: teste.org para api.teste.org.

Regra de nomes de subdomínio para Portais do desenvolvedor. Um domínio superior deve identificar seu proprietário e acrescentar um subdomínio para especificar o caminho Developer. Exemplo: teste.org para developer.teste.org.

## 6.4 MODELAGEM DE CÓDIGO

O caminho URI transmite modelo de recurso de uma API REST, com cada barra separando os objetos correspondentes ao segmento do caminho e a um recurso único dentro da hierarquia do modelo definido.

Modelagem de recursos é um exercício que estabelece os conceitos chaves da sua API no processo. Onde é semelhando à modelagem de dados para um esquema de banco de dados relacional ou modelagem clássica de um sistema orientado a objetos. Antes de ir diretamente ao ponto de URI, devemos parar e pensar no que pode ser útil como modelo de recurso API REST.

## 6.5 ARQUÉTIPOS DE RECURSOS

Como padrões de projeto, os arquétipos de recursos nos ajudam consistentemente na estrutura e no comportamento que são comumente encontrados em projetos de API REST. A API possui quatro tipos de arquétipos são: (document, collection, store e controller).

Document é um conceito singular que se assemelha a uma instância ou objeto de banco de dados. Representação de estado de um documento inclui ambos os campos com os valores e links para outros recursos relacionados. Exemplo de identificação do recurso do documento: http://localhost:8080/fabrica-tecido/tecido-vermelho.

Collection é um recurso que o cliente pode propor a adição de novos objetos na Collection e o recurso irá criar uma instancia ou não. Exemplo: http://localhost:8080/fabrica-carro/carros.

Store é um repositório gerenciado pelo cliente, um recurso de armazenamento permite os recursos HTTP para o cliente por conta própria. Exemplo: http://localhost:8080/loja-automovel/carros/123 é uma requisição PUT para alterar o carro 123.

Controller é um modelo de recursos procedural, onde são como executáveis de funções com parâmetros e valores de retorno, entrada e saída. Exemplo: http://localhost:8080/fabrica-chocolate/chocolates/1.

## 6.6 DESIGNER DOS CAMINHOS URI

Cada segmento de caminho URI, separados por barras representa uma oportunidade de Designer. Atribuindo valores significativos para cada segmento de caminho ajuda a comunicar de forma clara a estrutura do projeto.

Regra que deve ser usada um substantivo singular para nomes dos documentos. Exemplo: http://localhost:8080/funcionarios/arquitetos-software/paulo.

Regra um substantivo no plural deve ser usado para nomes de coleções. Exemplo: http://localhost:8080/jogos/dungeons-dragons/jogadores.

Regra um substantivo no plural deve-se armazenar nomes no URI para identificação do recurso. Exemplo: http://localhost:8080/musicas/david-guetta/playlists errado seria http://localhost:8080/musicas/david-guetta/playlist, pois estou com ujm recurso que é uma coleção de dados.

Regra de expressão verbal ou verbos devem ser usados como nomes de Controller, neste caso um URI deve identificar o recurso passado para a Controller e ser nomeado para indicar sua ação. Exemplo: http://localhost:8080/estudante/paulo/registrar.

Regra os segmentos de variáveis podem ser substituídos com base em identidades e valores, alguns caminhos podem ser estáticos e outros segmentos de URI são variável o que significa que são automaticamente preenchidos e fornece a singularidade do URI. E uma substituição de um URI pode ser feito a partir da API REST escolhida e assim que o cliente executar a substituição, pode ser repassado um identificador numérico ou alfanumérico. Exemplo: http://localhost:8080/escola-municipal/turma-7/aluno/12 ou http://localhost:8080/escola-municipal/turma-7/aluno/1ab2.

Regra de nomes de funções CRUD não deve ser usado em um URI, a utilização da semântica do URI deve ser apenas para identificar recursos e valores variável. Exemplo: http:localhost:8080/ctis/ms/colaboradores/123 dessa forma está correto, errado seria a utilização dessa forma http://localhost:8080/ctis/ms/atualizar-colaboradores/123, além de ser fora do contexto HTTP, está mostrando qual funcionalidade do CRUD irá executar deixando frágil seu código.

## 6.7 DESIGNER DE CONSULTA

Designer de consulta fornece regras relativas à concepção de URI. Referente a RFC3986 é opcional este tipo de consulta em URI onde vem após seus fragmentos, exemplo ( URI=esquema"://"autoridade"/"caminho[“?”consulta][“#”fragmento] ). Como um componente a consulta serve para a busca de um fragmento passado e retorna apenas um resultado.

Regra de componente de consulta pode ser usada com um conjunto de filtros, cada componente da consulta é um fornecimento de critérios para a pesquisa. E uma

representação do estado de mensagens de entrada e saída. Exemplo: http://Iocalhost:8080/ctis/colaboradores/name=Paulo usando um GET para retorno da busca.

Regra de paginação da consulta deve ser usada usando os princípios do Designer de Consultas, deve ser passada como parâmetro na coleção e assim a paginação sendo executada por demanda. Exemplo: http://Iocalhost:8080/pessoas?pageSize=2&pageStaterindex=50.

## 6.8 MÉTODOS DE REQUISIÇÃO

A RFC2616 Request-Line trabalha com o URI e os métodos específicos do HTTP 1.1 e cada método HTTP tem a semântica especifica e definida para o seu funcionamento e contexto.

Regra os métodos GET e POST não devem ser usados fora de seu contexto, assim foge de toda a intenção da mensagem e máscara o propósito do método HTTP.

Regra GET deve ser utilizada para recuperar dados de uma consulta qualquer, apenas isso.

Regra HEAD deve ser utilizado para recuperar os cabeçalhos de uma requisição/resposta, o HEAD recebe os dados como o GET mas sem um corpo e este recurso a usado para recuperar os metadados de uma requisição.

Regra o PUT deve ser usado para atualizar ou inserir um dado a partir de um objeto jogado na URI de chamada.

Regra o PUT deve ser utilizado para modificar/atualizar um dado mutável.

Regra POST deve ser utilizada para criar um item na coleção.

Regra POST deve ser usada para invocar controladores orientados por função. Regra DELETE deve ser usado para deletar/apagar um registro a partir de uma requisição feita pelo usuário.

Regra OPTIONS deve ser usada para recuperar metadados que gerou uma iteração e está disponível no cabeçalho de conversação.

## 6.9 CÓDIGO DOS STATUS DE RESPOSTA

REST API usa o estado de respostas requisitadas pelos clientes, forma de códigos HTTP response. Segue abaixo a descrição de cada um e forma de utilização.

| |
|---|
| Código 200 indica que sua requisição foi um sucesso, todo sucesso no código fonte deve-se retornar HTTP 200. |
| Código 201 ele deve ser a resposta de inserção de algum dado feito pelo Controller ou Collection. |

| |
|---|
| Código 202 enviado para o Controller que obteve uma conexão assíncrona. |
| Código 204 a resposta ira mostra que intencionalmente o corpo de código foi deixado em branco. |
| Código 301 indica que urn novo URI foi inserido para o cliente. |
| Código 303 resposta a enviado para retornar dados que considera opcional. |
| Código 304 resposta enviada para preservar o consumo da banda. |
| Código 307 indica que uma URI foi gerado temporariamente para o cliente consumir seu recurso. |
| Código 400 indica que o caminho acessado está com erro. |
| Código 401 indica erro que o cliente não está autorizado para o envio da requisição. |
| Código 402 envia para a requisic5o negar o recurso para o cliente. |
| Código 404 a requisicao pode estar fora ou nao existe. |
| Código 405 interage corn o cliente a partir do momento que o cliente envia urn método HTTP nao suportado. |
| Código 406 erro retornado quando o cliente tenta solicitar tipo de dados não suportado. |
| Código 409 indica que o cliente tentou violar urn estado do objeto. |
| Código 412 mostra para o cliente que uma de suas condicties requisitadas nao foi cumprida. |
| Código 415 erro mostrado quando o cliente envia dados de uma mídia nao suportados. |
| Código 500 retorna para o cliente dizendo que possui erro grave. |

### 6.10 API´S REST

A arquitetura MME disponibiliza as API's para Java e PHP. Java é utilizado o Resteasy e o PHP são utilizados o Symfony e o Slim. Para maiores detalhamentos executar a leitura do documento de Diretriz Arquitetural.

## 7. CONCLUSÃO

Um padrão unificado de código mantém o código legível e com facilidade de manutenção. Os nomes significativos deixam a leitura do código simples e entendível.

De acordo com o documento, facilitara para todos os desenvolvedores a escrita e a forma de trabalhar no projeto, beneficiando um padrão mínimo, e, desta forma, uma conduta mínima de validação do código.

Conclui-se que o desenvolvimento seguirá uma linha de processo de produção padronizado e com qualidade, ajudando o andamento do projeto e com a não refatoração de código fonte.

## 8. REFERÊNCIAS


- Manual de nomenclature Java — RUP — Rational IBM
- Codigo Limpo "Habilidades Praticas do Agile Software" — Edicao Revisada — Robert C. Martin.
- REST Construa API's inteligentes de maneira simples — Alexandre Saudate
- REST API — Mark Masse
- JavaScript Designer Patterns — Addy Osmani
- Sun Certified Programmer for Java 6 Study Guide — Kathy Sierra e Bert Bates
- http://www.php-fig.org/ - Acessado 15/04/2024