Padronização das aplicações em Node.js

Solicitante:	
Responsável:	Kennedy Santos

Introdução

Este documento descreve algumas definições técnicas, conceituais e gerencias para a padronização das aplicações.

Conceitos básicos

Seguem os ajustes gerais, necessários para padronização e adequação dos projetos do Visão 360 ao novo fluxo DevOps.

- Os microserviços da 360 são especializados em acessar e retornar informações que existem no Cluster Cloudera. Logo estes tem a capacidade de acessar ferramentas como HBase, Impala e Hive

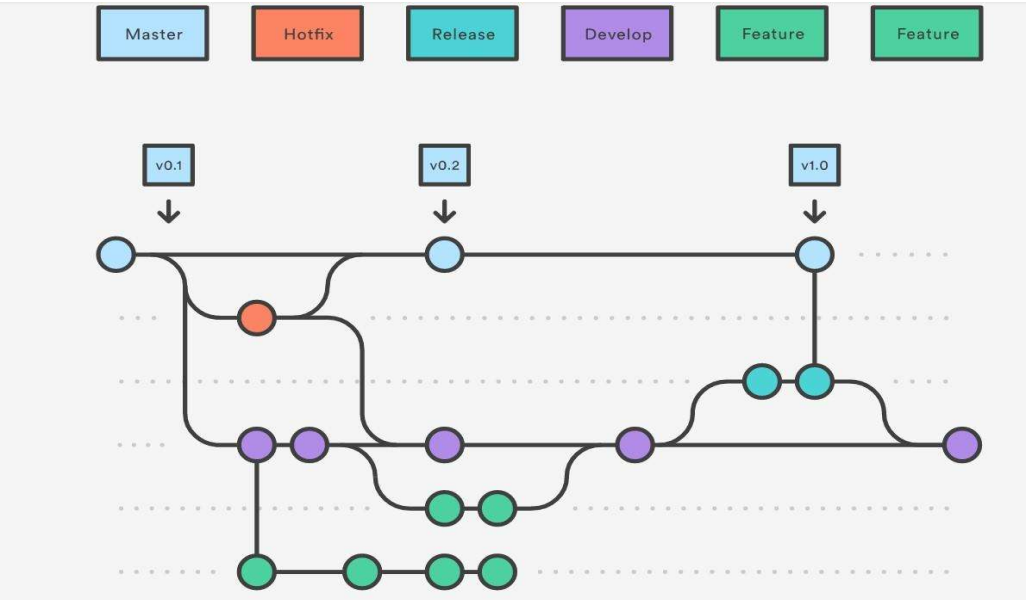
Frameworks

Aqui estão descritos todos os artefatos de software que entregam funcionalidades encapsulada para os desenvolvedores de modo a esconder complexidades, economizar tempo de desenvolvimento, fazer injeção de comportamento e etc.

- Express
- HBase-rpc-client
- Eureka

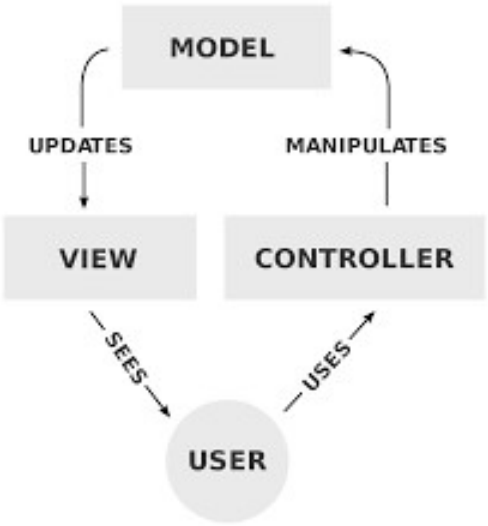
- **GITFLOW workflow**

Toda alteração deve ser realizada a partir da branch dev criando uma nova branch denominada feature/devOps após a finalização deve ser feito o merge com a branch dev novamente.



Padrão de camadas

Por ser uma Aplicação WEB, os Micro Serviços utilizam o padrão MVC de camadas. As Camadas visam facilitar o desenvolvimento, entendimento e organização do código, mantendo baixo acoplamento e alta coesão dos mesmos.



Seguem abaixo as camadas definidas para os micro serviços desta LPS.

- **Model**
    - Define o(s) modelo(s) do dado retornado de uma fonte como um Objeto daquela linguagem.
- **Service**
    - Regras de negócio do serviço.
    - Transformação do dado da Base ou de outra fonte para que melhor se encaixe no que foi definido o escopo do Serviço.
        - Ex: traduzir campos com tabelas de domínio.
- **Factory**
    - Conexão com fontes externas. Funções de **CRUD.**
    - As fontes podem ser Bancos de dado Sql, NoSql, Arquivos.
- **Controller**
    - Porta de entrada e saída da API
    - Valida requisições e parâmetros, direcionando para a camada de serviço correta.
    - Prepara e retorna **Responses.**
- **Routes**
    - Camada que define as rotas, parâmetros e métodos HTTP para acesso das Controllers.
- **Utils**
    - Mantém classes utilitárias com o Tranformação de datas, Handlers de configuração.

- **Padrão de requisição REST**

Das etapas de padronização

Atividades a serem realizadas:

1. Ajustar a estrutura do projeto para fins de padronização
2. Aplicar os testes unitários
3. Aplicar conformidade com o "**ESLint**"

Detalhes de padrões da estrutura do projeto:

O repositório deverá conter a seguinte estrutura, considerando o que é usado ou não pela aplicação (ajustar nomenclatura e apontamento dos diretórios se necessário):

```
.
├── src
│   ├── config
│   │   ├── resources
│   │   └── data.yml
│   ├── controllers
│   ├── exceptions
│   ├── graphql
│   ├── utils
│   ├── models
│   ├── factorys
│   ├── routes
│   └── services
├── test
├── .gitignore
├── .npmignore
├── README.md
└── package.json
└── server.js
```

*****Remover arquivos desnecessários (adicionar ao .gitignore se for preciso):**

Da aplicação dos testes unitários:

- Deve ser realizado o teste unitário de cada função/componente que estiver dentro da arquitetura.

Configurações de conformidade com o DevOps e ESLint:

- **Variáveis (".env", ".npmrc_config", "resources/config/*yml") -** Usar process em "/src/config/index.js", os valores estarão configurados no "S.O." de acordo com as variáveis definidas no OpenShift.
- **Jenkinsfile -** Remover
- **.nyc_output -** Não versionar
- **docs/swagger.json -** Não versionar
- Não utilizar o subdiretório "**main**" após o "src".

- **eslint.yml** - Não versionar, processo executado via pipeline - ALM aplica os ajustes. Segue exemplo de arquivo lint para configuração local:

```
{
  "env": {
      "browser": true,
      "commonjs": true,
      "es2020": true
  },
  "extends": "strongloop",
  "parserOptions": {
      "ecmaVersion": 11
  },
  "rules": {

  }
}
```

- **Padronizar e atualizar os apontamentos do "package.json" conforme nova estrutura. Obs.: Considerar as configurações "devDependencies", "registry" e desconsiderar o "lint".** Exemplo:

```
{
  "name": "nodejs-rest-app",
  "version": "1.0.0",
  "description": "aplicação exemplo para utilização de REST",
  "main": "src/app.js",
  "scripts": {
    "lint": "eslint src --fix",
    "test": "nyc --reporter=lcov --reporter=text-lcov mocha --
exit=true",
    "start": "nodemon"
  },
  "repository": {
    "type": "git",
    "url":
"http://git.caixaseguros.intranet/corporativo/archetypes/nodejs-rest-
app.git"
  },
  "publishConfig": {
    "registry":
"http://nexus.caixaseguros.intranet/nexus3/repository/npm-internal/"
  },
  "author": "Marcelo Nogueira
<marcelo.nogueira.altran@caixaseguradora.com.br>",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "method-override": "^3.0.0",
    "reflect-metadata": "^0.1.13",
    "winston": "^3.3.3",
    "winston3-logstash-transport": "^1.0.1-c"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "eslint": "^7.6.0",
    "eslint-config-strongloop": "^2.1.0",
    "mocha": "^8.1.1",
    "nodemon": "^2.0.4",
    "nyc": "^15.1.0",
    "sinon": "^9.0.3",
```

```
    "supertest": "^4.0.2"
  }
}
```