Ibama

Arquitetura de Sistemas

Documento de Arquitetura de Sistemas

Versão 1.2

Histórico da Revisão

Data	Versão	Descrição	Autor
26/04/2018	1.0	Criação do documento para a versão 1.0 do framework com base em referências do ePing e eMag.	Flaviano O. Silva
05/06/2018	1.1	Adicionadas recomendações para Arquitetura híbridas e WEB, navegadores e sistemas operacionais *mobiles* que devem funcionar o sistema.	Flaviano O. Silva
20/03/2019	1.2	Inclusão do item servidores web (legado).	Flaviano O. Silva

Índice de Figuras

Figura 1 - Situação atual dos Sistemas ..... **Erro! Indicador não definido.**
Figura 2 - Arquitetura WEB, Infraestrutura e Legado 8
Figura 3 - Arquitetura em Alto Nível - híbrida e web 16
Figura 4 - Arquitetura em Camadas Híbrida 18
Figura 5 - Arquitetura Alto Nível - WEB 19
Figura 6 - Camada de Apresentação 20
Figura 7 - Camada Apresentação WEB 22
Figura 8 - Camadas SpringBoot 23
Figura 9 - Cobertura de Qualidade - Sonar 27
Figura 10 - Exemplo de Dashboard SonarQube 28
Figura 11 - Exemplo de Dashboard detalhado de um projeto 29
Figura 12 - Repositório de Trabalho 30
Figura 13 - Tags Javadoc 34
Figura 14 - Documentação Javadoc 35

Índice Analítico

1. INTRODUÇÃO 5

2. REPRESENTAÇÃO ARQUITETURAL 13

2.1 ARQUITETURA HÍBRIDA (WEB E MOBILE) 13
2.2 ARQUITETURA HÍBRIDA (WEB) 14
2.3 CAMADA VIEW 19
2.3.1 Camada VIEW - Híbrida 19
2.3.2 Camada VIEW - WEB 20
2.4 CAMADA API GATEWAY 21
2.5 CAMADA BACKEND (SPRINGBOOT) 22
2.5.1 Camada Resouce 22
2.5.2 Camada Service 22
2.5.3 Camada Repository 23
2.5.4 Camada Model 23

3. SERVIDOR DE APLICAÇÃO 23

4. NAVEGADORES 24

5. SISTEMAS OPERACIONAIS MÓVEIS 24

6. METAS E RESTRIÇÕES DA ARQUITETURA 25

7. QUALIDADE 25

8. DISTRIBUIÇÃO 30

9. FERRAMENTAS E TECNOLOGIAS 31

10. DOCUMENTAÇÃO DOS CÓDIGOS FONTES 32

11. TREINAMENTOS 33

12. REFERÊNCIAS 37

1. INTRODUÇÃO

De acordo com FILHO, 2013, software, de modo genérico, é uma entidade que se encontra em quase constante estado de mudança. As mudanças ocorrem por necessidade de corrigir erros existentes no software ou de adicionar novos recursos e funcionalidades.

A ausência da padronização de arquitetura de sistemas deixa a área de desenvolvimento receptível a vários modelos arquiteturais utilizados por cada prestador de serviço ou até mesmo por áreas internas da instituição. Este cenário dificulta, assim, a implantação de melhorias contínuas, o aperfeiçoamento do aprendizado técnico especializado, além de se comprometer o repasse de conhecimento referente às várias arquiteturas utilizadas, além disso dificulta a Governança Tecnológica, conforme observado na figura 01.

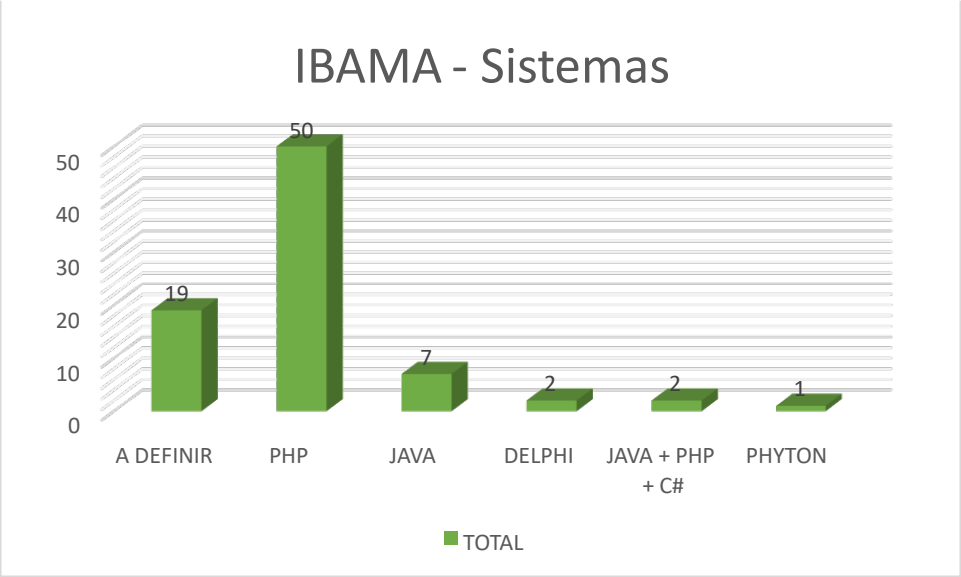

Figura 1 *- Situação atual dos Sistemas*

Ao se identificar tal fragilidade e sabendo da importância de se ter uma arquitetura padrão, a CGTI juntamente com a coordenação de sistemas realizou estudos, pesquisas

e análises técnicas em prol de uma arquitetura robusta, segura, que possui longevidade, simplicidade no seu uso e levando como ponto principal a observância em atender o negócio do IBAMA que se apresenta em um cenário de crescente demanda por soluções negociais automatizadas, tendo em vista a amplitude do IBAMA e do Ministério do Meio Ambiente (MMA).

Deste modo, a área de TI do IBAMA a partir desse momento passa a adotar como padrão para os desenvolvimentos dos sistemas de informação a arquitetura de referência, onde será adotado as tecnologias e padrões de desenvolvimento para todos os tipos de aplicações, incluindo aplicações Mobile.

Essa arquitetura trará ganhos em termos de padronização, reuso, manutenabilidade, escalabilidade, velocidade no desenvolvimento dos sistemas de informação, portabilidade, dentre outras características, além de permitir a implantação dos padrões de controle de acessibilidade do Governo Federal (e-PING e e-MAG).

> *A arquitetura ePING – Padrões de Interoperabilidade de Governo Eletrônico – define um conjunto mínimo de premissas, políticas e especificações técnicas que regulamentam a utilização da Tecnologia de Informação e Comunicação (TIC) no governo federal, estabelecendo as condições de interação com os demais Poderes e esferas de governo e com a sociedade em geral. Portal de Governo Eletrônico do Brasil, 2014.*
>
> *O Modelo de Acessibilidade em Governo Eletrônico (**eMAG**) consiste em um conjunto de recomendações a ser considerado para que o processo de acessibilidade dos sítios e portais do governo brasileiro seja conduzido de forma padronizada e de fácil implementação. Portal de Governo Eletrônico do Brasil, 2014.*

Ademais para a essa nova Arquitetura todos os componentes e bibliotecas necessários ao desenvolvimento são totalmente sem ônus ao IBAMA, além de ser uma

arquitetura desacoplada sempre priorizando soluções desenvolvidas em Software Livre, conforme IN-04 de Setembro de 2014, Capítulo II, Subseção IV das letras c, d e e

> *c) a capacidade e alternativas do mercado, inclusive a existência de software livre ou software público;*
>
> *d) a observância às políticas, premissas e especificações técnicas definidas pelos Padrões de Interoperabilidade de Governo Eletrônico - e-PING e Modelo de Acessibilidade em Governo Eletrônico - e-MAG, conforme as Portarias Normativas SLTI nº 5, de 14 de julho de 2005 e nº 3, de 7 de maio de 2007; e) a aderência às regulamentações da Infraestrutura de Chaves Públicas Brasileira - ICPBrasil, conforme a Medida Provisória nº 2.200-2, de 24 de agosto de 2001, quando houver necessidade de utilização de certificação digital; IN-04, atualiza em Setembro de 2014.*

A TI acredita no seu padrão arquitetural adotado para os sistemas e fortalece o compromisso com áreas de negócio do IBAMA em entregar seus produtos com maior qualidade, menor prazo e utilizando as melhores tecnologias de desenvolvimento de sistemas do mercado de forma a agregar valor significativo ao negócio. É possível observar na figura 01, que existem no momento 19 sistemas aguardando a inicialização dos trabalhos já baseadas na nova Arquitetura.

Com isso, é importante salientar que à medida que tamanho e complexidade dos sistemas de software aumentam, o problema de projeto extrapola as estruturas de dados e algoritmos de computação. Ou seja, projetar a arquitetura (ou estrutura geral) do sistema emerge como um problema novo. Questões arquiteturais englobam organização e estrutura geral de controle, protocolos de comunicação, sincronização, alocação de funcionalidade a componentes e seleção de alternativas de projeto. Por exemplo, nos sistemas Web, uma solução que tem sido empregada faz uso de múltiplas camadas separando componentes cliente, servidores de aplicações, servidores Web e outras aplicações (que possam ter acesso a esse sistema), como mostra a figura 2. Essa

estruturação em camadas objetiva facilitar a alocação da funcionalidade aos componentes. O uso de camadas oferece suporte à flexibilidade e portabilidade, o que resulta em facilidade de manutenção.

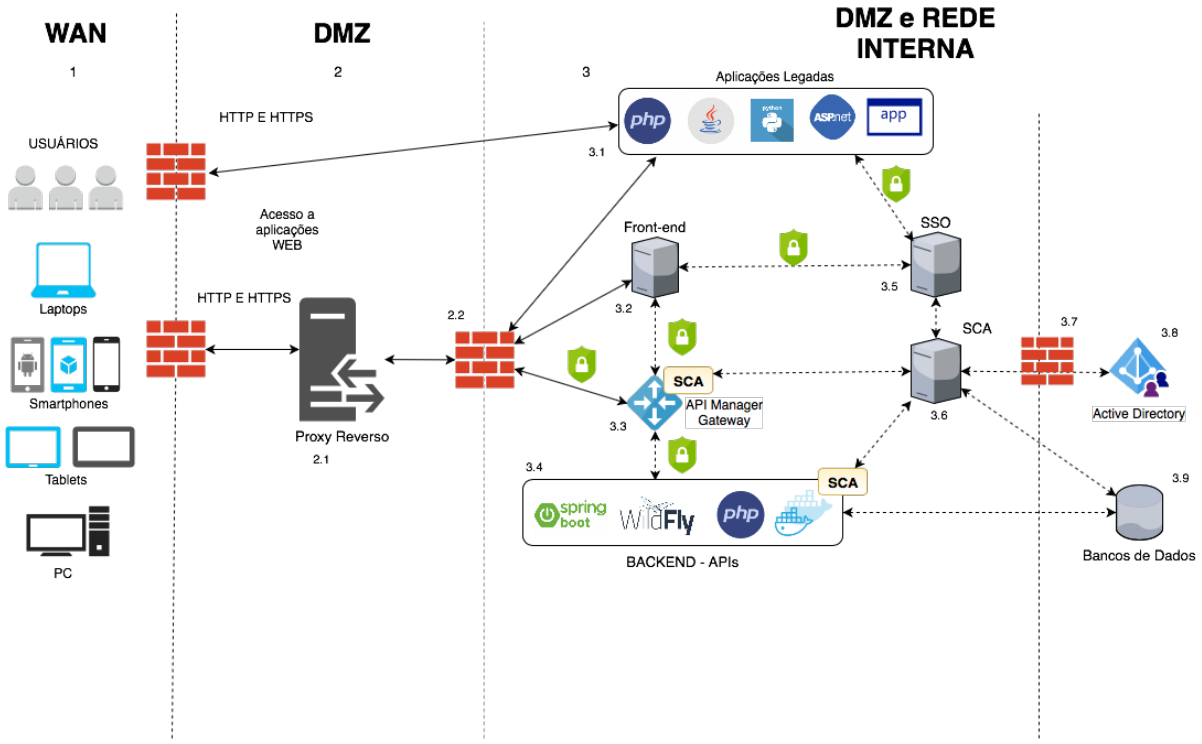

Figura 2 *- Arquitetura WEB, Infraestrutura e Legado*

1.1) Arquitetura WEB, infraestrutura e Legado:

Conforme a figura acima a arquitetura faz uma separação de responsabilidades com foco em segurança e proteção do ambiente interno.

1. **WAN:** à Rede externa ao IBAMA de onde serão executados os acessos, no caso de usuários da rede interna o entendimento é o mesmo. O acesso pode ser provido por qualquer dispositivo de acordo com a aplicação que se deseja utilizar.

2. **DMZ:** A DMZ ou rede desmilitarizada deve possuir apenas um proxy reverso para sistemas novos, os sistemas legados são liberados via regras de firewall.
   - **2.1. Proxy reverso:** Além da proteção dos servidores da rede interna fará o encaminhamento para o servidor solicitado ou o balanceamento para outros nós (servidores) conforme necessidade da aplicação.

     O acesso também poderá ser à uma API, como será o caso das aplicações mobile, que farão o acesso diretamente ao Gateway de API (3.3), este mesmo cenário pode ocorrer para aplicações legadas, dependendo apenas na forma que foi concebida.
   - **2.2. Firewall DMZ x Rede Interna:** Este firewall tem como premissa proteger o acesso aos servidores internos, sendo que só serão liberadas as portas dos serviços das aplicações que o proxy reverso necessita.
3. **Rede Interna:** A Rede interna é onde se encontram os servidores que devem ser protegidos e que possuem os serviços WEB necessários.
   - **3.1. Aplicações Legadas:** As aplicações legadas são as aplicações já existentes dentro da Instituição que podem ou não serem evoluídas ou até mesmo migradas para a nova arquitetura, para isso é necessário um estudo de migração. Esta padronização visa atender a evolução do IBAMA porém sabe-se da necessidade da existência das aplicações legadas dentro do ecossistema. As aplicações devem seguir a mesma ideia de funcionar atrás de um proxy reverso (sempre que possível).
     - 3.1.1. **Servidores WEB - Java**: O servidor de aplicação definido internamente para os sistemas legado utilizando tecnologia Java é Wildfly 8 ou superior (*community edition* do JBoss EAP Red Hat), sendo atualizado de acordo com a necessidade e definido pelo arquiteto. Para mitigar os riscos de atualizações de versões de servidor

e componentes, a premissa é que todos as tecnologias utilizadas sigam a especificação Java EE 6 ou superior e não sejam adotados componentes proprietários.

O Wildfly Application Server (ou, abreviadamente, Wildfly) é um servidor de aplicação de código aberto desenvolvido pela JBoss (atualmente, uma divisão da Red Hat) para a plataforma Java EE. Ele oferece toda a infraestrutura necessária para executar aplicações Web desenvolvidas sobre essa plataforma, além disso, é completamente compatível com especificação Java EE 6 (diz-se “Java EE 6 Full Profile”), tendo recebido inclusive certificação da Oracle. Assim, ao executar as aplicações Web nesse servidor, se elas foram desenvolvidas obedecendo aos padrões da plataforma, teremos a certeza de que elas funcionarão como desejamos, é claro que eventualmente erros de lógica podem ser cometidos por programadores, o fato de uma aplicação atender à especificação não garante que ela esteja livre desses erros.

3.1.2. **Servidor WEB – PHP:** Segundo a Apache Foundation, o Apache HTTP Server é um servidor de código aberto para SO Unix/Windows mantido pela Apache Software Foundation.

Apache HTTP Server é uma das tecnologias de servidores web mais antigas e confiáveis, além disso, possui mais de 20 anos de mercado, sempre com grandes evoluções.

Para que seja utilizado nas aplicações legadas desenvolvidas em PHP são necessários a inclusão de módulos correspondentes à esta tecnologia, o APACHE utilizado deve ser 2.4 ou superior, as aplicações que necessitarem de versões anteriores devem ser ajustadas para

garantir que estão utilizando o pacote mais atualizado evitando ataques e inconsistências.

3.2. **Front-end:** O front-end representa as novas aplicações WEB essas aplicações são desenvolvidas seguindo a ideia de separação da estrutura mais robusta, tornando-se aplicações mais leves e com um processamento maior pelo lado do cliente, além disso tornasse mais escalável e com um tempo de resposta melhor. Dentro das evoluções tecnologias uma aplicação que necessita apenas de um browser (navegador) para funcionar torna-se muita mais disponível para um cliente.

3.3. **API Manager (Gateway):** O Gestor de APIs é um dos componentes mais importantes da nova Arquitetura e nova estratégia para desenvolvimento de software, tem como função a gestão de todas as APIs disponibilizadas pelo órgão, mesmo internas, onde será possível garantir a segurança de acessos através do SCA – Sistema de Controle de Acesso e obter informações estratégicas de uso de API possibilitando uma governa para esse ativo, com possibilidade de tomadas de decisões para melhorias das APIs de forma contínua.

3.4. **Back-end das APIs:** O Backend sem dúvida é a parte mais importante referente ao novo paradigma de desenvolvimento em camadas, ele é a camada que necessita ser mais robusta, escalável e que irá consumir mais recursos computacionais. As novas APIs devem ser desenvolvidas em servidores autocontido ou com funcionamento em tecnologias baseadas em containers. A ideia de um *microserviços* é sempre a independência de outras estruturas para que o mesmo fica com a possibilidade de escala e processamento mais rápido que tecnologias anteriores.

3.5. **SSO:** O Single Sin On é a garantia de um login único para o usuário utilizar em várias aplicações, essa estrutura já existe e faz parte da arquitetura do SCA já adotada pelo órgão, desta forma, as novas aplicações devem utilizar a mesma forma quando possível. No caso de aplicações WEB essa adoção é obrigatória já no caso de aplicações mobile serão acessados APIs que consomem informações do SCA para ter esse tipo de acesso.

3.6. **SCA:** O Sistema de Controle de Acessos do IBAMA é responsável pelo controle e auditoria dos acessos aos sistemas do órgão além disso os usuários externos também estão cadastrados no SCA através da base de dados, na nova arquitetura isso não poderá ser modificado sendo que o SCA será um componente importante para a segurança das APIs construídas.

3.7. **Firewall x Active Directory e Banco de dados:** Apesar dos servidores estarem na rede interna o servidor de usuários e o banco de dados possuem um acesso controlado, o firewall deve liberar quais servidores podem ter acesso ao Active Directory e aos servidores de Bancos de Dados.

3.8. **Active Directory:** Estrutura de usuários do órgão, todos os servidores estão cadastrados no Active Directory e tem o seu mapeamento no SCA para terem acessos aos sistemas do IBAMA.

3.9. **Banco de dados:** É representado no desenho como toda a estrutura de banco de dados, independente de tecnologia, o órgão já possui bancos de dados Oracle, porém para nova arquitetura a tecnologia não tem muita relevância, mais para o órgão o banco de dados Oracle está totalmente acoplado aos sistemas legado, isso ocorre devido as rotinas dos sistemas estarem definidas na sua grande maioria como Store Pr*ocedures do banc*o.

1.2) Importância da Arquitetura de software

Todos esses fatores compreendem o projeto no nível arquitetural e estão diretamente relacionados com a organização do sistema e, portanto, afetam os atributos de qualidade (requisitos não funcionais) como desempenho, portabilidade, confiabilidade, disponibilidade, entre outros. Se for feita uma comparação entre arquitetura de software (caracterizada, por exemplo, pelo estilo em camadas) e arquitetura "clássica" (relativa à construção de edificações), podese observar que o projeto arquitetural é determinante para o sucesso do sistema.

A arquitetura proposta tem como principais objetivos: longevidade, simplicidade na evolução, escalabilidade e velocidade no seu desenvolvimento.

1.3) Finalidade

O objetivo deste documento é a definição da arquitetura de software para o desenvolvimento de novos sistemas do IBAMA. Derivações desta arquitetura deveram ser feitas sob demanda, de acordo com as necessidades do projeto, pela equipe de Arquitetura.

1.4) Visão Geral

> *Em ciência da computação, front-end e back-end são termos generalizados que se referem às etapas inicial e final de um processo. O front-end é responsável por coletar a entrada do usuário em várias formas e processá-la para adequá-la a uma especificação em que o back-end possa utilizar. KA PING, 2003.*

A presente arquitetura divide toda aplicação Web em dois sistemas: cliente e servidor. O sistema cliente é organizado de maneira a possibilitar integração com diversos serviços de maneira simples e rápida. O sistema servidor é organizado de forma a atender

qualquer tipo de aplicativo cliente, de maneira a possibilitar a integração de sistemas desktop, móveis ou web (novos ou legados). A comunição entre os sistemas cliente e servidor será feita usando o protocolo comunicação HTTP seguindo a arquitetura de serviços RESTful. O Ideal que essa comunicação ocorra através de um Gateway de APIs, desta forma será feito o controle no que diz respeito principalmente a segurança e quantidade de acesso o Gateway será responsável por mediar a comunicação entre os consumidores (sistemas) e os provedores (back-end).

2. Representação Arquitetural

2.1 Arquitetura Híbrida (WEB e Mobile)

Esta arquitetura tem como premissa o aproveitamento de código-fonte e evitar o retrabalho com o desenvolvimento de dois projetos para atender a mesma finalidade, desta forma a arquitetura proposta é composta pelas seguintes camadas principais:

Tecnologia	Descrição
FRONT-END	
IONIC	Camada de apresentação (HTML5, CSS3 e JavaScript).
AngularJS	Camada de Integração com Backend.
FRONT-END (MOBILE)	
IONIC	Camada de apresentação (HTML5, CSS3 e JavaScript), baseada em Cordova.
AngularJS	Camada de Integração com Backend.
BACK-END	
Zull	API Gateway
SpringBoot	Responsável pela construção de serviços RESTFul baseada em *microserviço* (altamente escalável).
Undertown	Servidor de aplicação autocontido;

Persistência	JPA, Hibernate ou JDBC.
Objetos de transporte	DTOs.

2.2 Arquitetura Híbrida (WEB)

Esta arquitetura é bem convergente com a arquitetura híbrida (web e mobile) porém sem a necessidade do framework IONIC (*cordova*) que faz a interface entre os sistemas operacionais mobiles e a aplicação. Desta forma a arquitetura proposta é composta pelas seguintes camadas principais:

Tecnologia	Descrição
FRONT-END	
WEB - W3C	Camada de apresentação (HTML5, CSS3 e JavaScript).
AngularJS	Camada de Integração com Backend.
BACK-END	
Zull	API Gateway
SpringBoot	Responsável pela construção de serviços RESTFul baseada em *microserviço* (altamente escalável).
Undertown	Servidor de aplicação autocontido;
Persistência	JPA, Hibernate ou JDBC.
Objetos de transporte	DTOs.

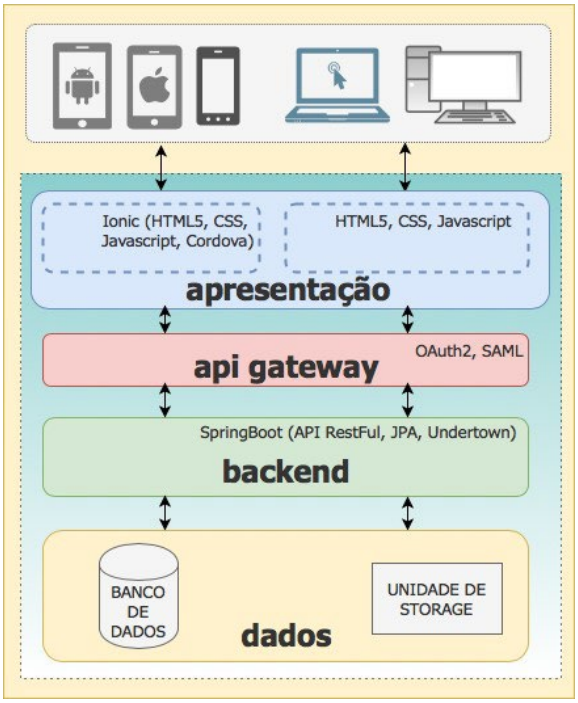

***Figura 3** - Arquitetura em Alto Nível - híbrida e web*

Conforme a figura 3 é possível perceber a divisão das responsabilidades, desta forma, cada camada tem seu papel bem definido. A camada de apresentação é responsável por expor o sistema para qualquer dispositivo, de acordo com a necessidade, esta camada ainda possui uma divisão dependendo do tipo de projeto:

- **Híbrido**: possuí as tecnologias necessárias para atender o desenvolvimento de aplicativos para dispositivos móveis (Android, iOS e Windows Phone), além disso o mesmo código fonte é utilizado na camada WEB acessada por navegadores.
- **WEB**: é apenas para funcionamento através de um navegador, ela deve ser responsiva para atender a necessidade de acesso via aparelhos móveis, porém a mesma não funciona com aplicativo das plataformas.

A camada do Gateway tem o papel de mediar toda a comunicação com o *backend* de forma controlada, em seguida, a camada backend possuí todos os *microserviços* necessários para os sistemas expostos como APIs REST e por fim a camada de dados fica responsável pela comunicação entre os bancos de dados, unidades de *storage* ou qualquer serviço externo.

Nas figuras 4 e 5 é possível observar o detalhamento e como ele se estende ao nível de componentes utilizados para prover a arquitetura, separando ainda em híbrido, web, *frontend* e *backend*.

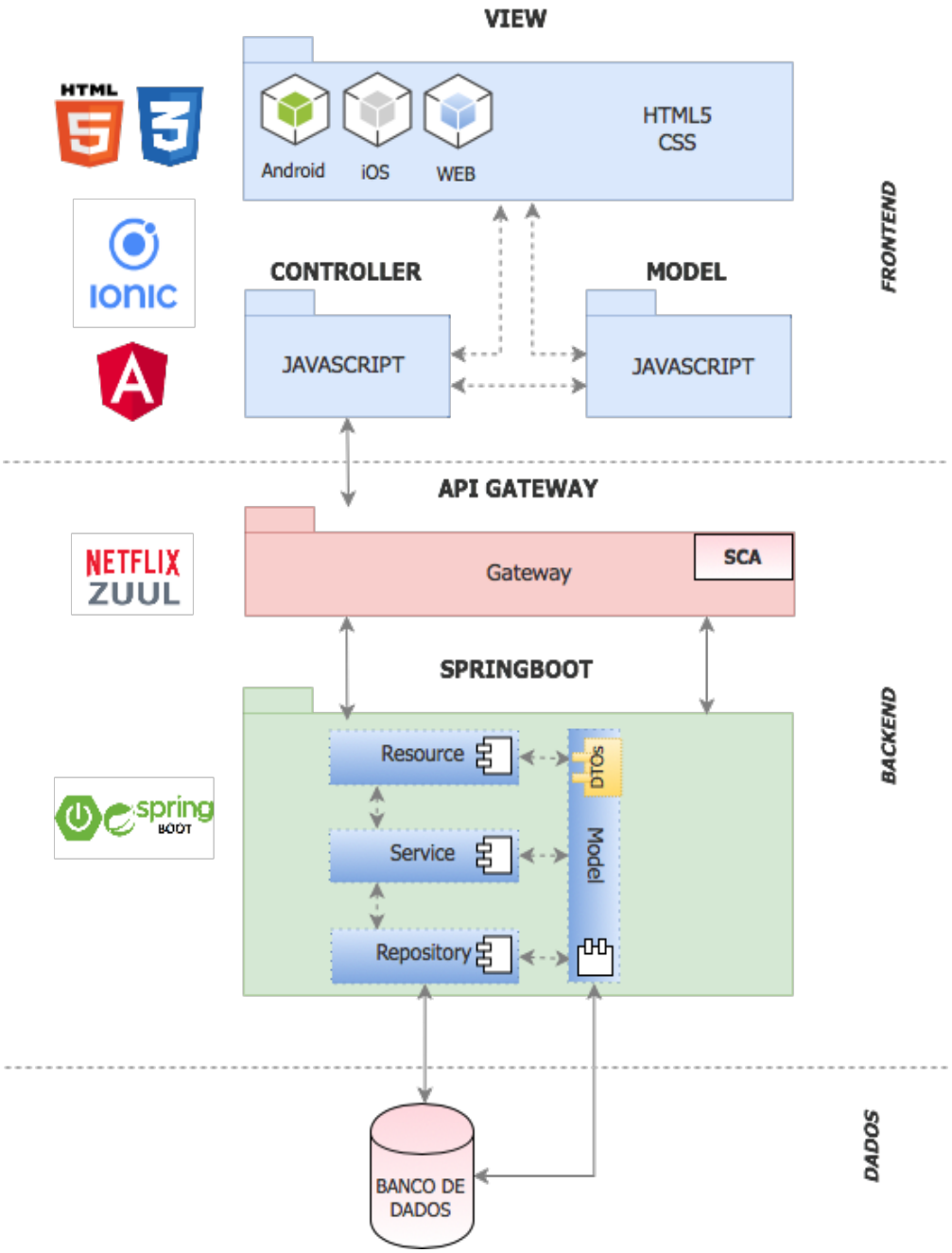

Figura 4 - Arquitetura em Camadas Híbrida

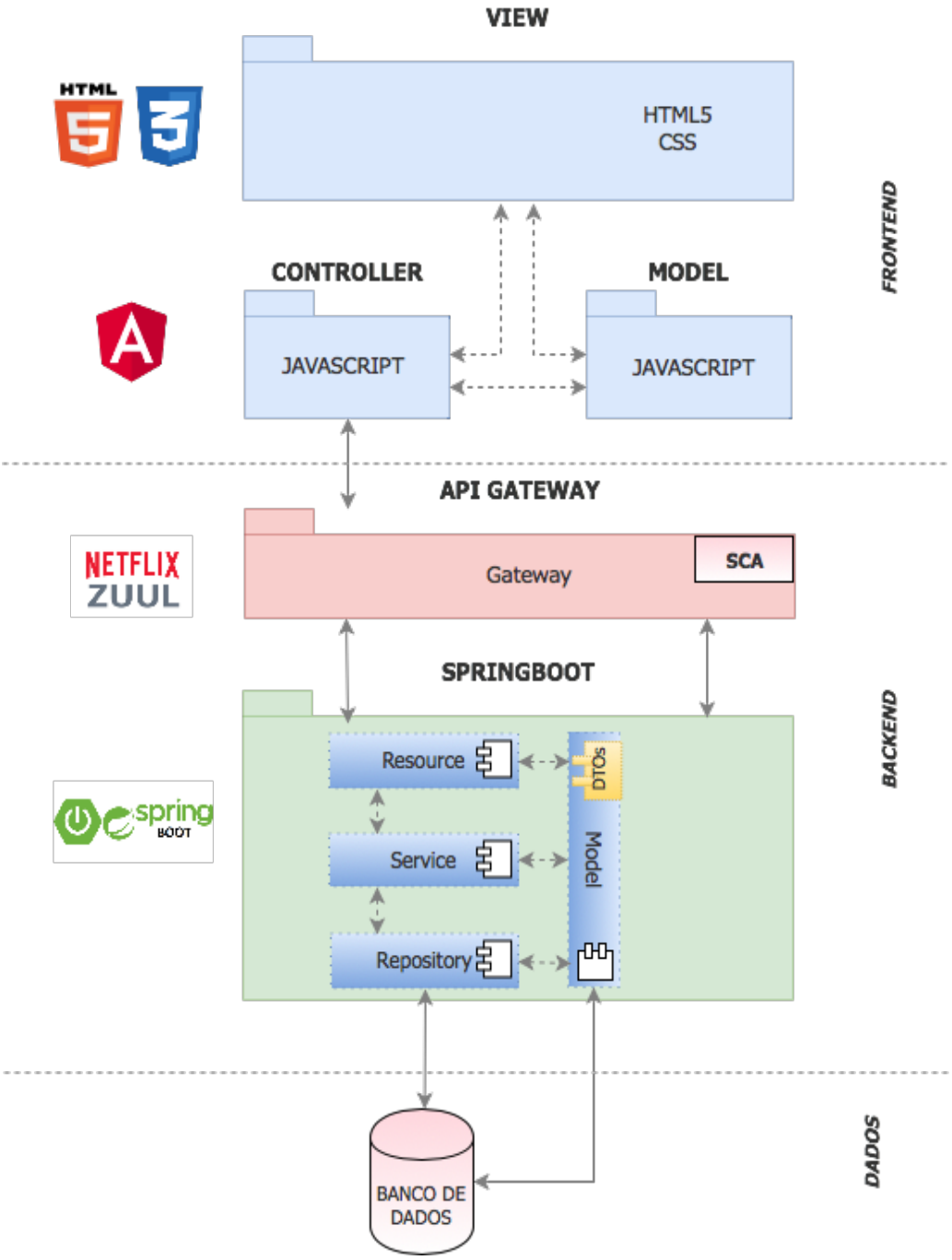

Figura 5 *- Arquitetura Alto Nível - WEB*

2.3 Camada VIEW

A camada VIEW deverá ser construída utilizando basicamente HTML5, CSS3 e JavaScript.

2.3.1 Camada VIEW - Híbrida

Para auxiliar seu desenvolvimento e aumentar a produtividade deverão ser utilizadas as bibliotecas jQuery e AngularJS suportado pelo IonicFramework que utiliza o Cordova como componente principal para mediação das aplicações mobile, tornado a aplicação híbrida. O AngularJS para controle de interação Model Presenter.

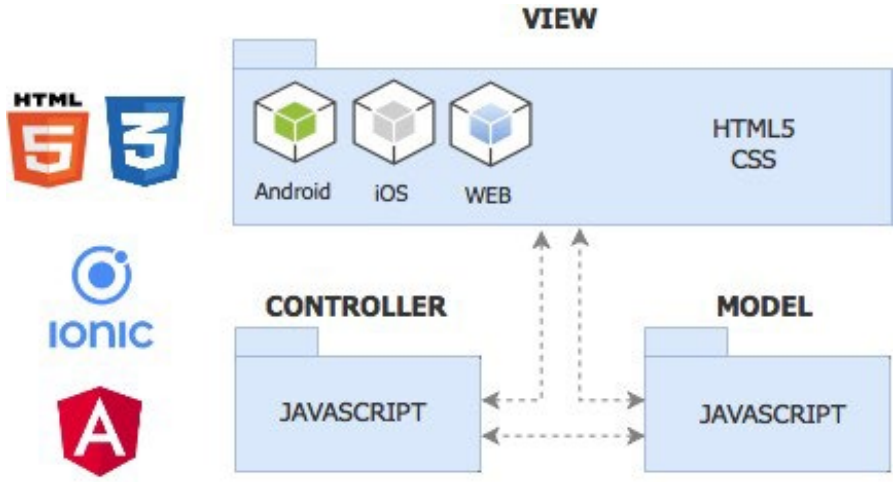

Figura 6 *- Camada de Apresentação*

O Ionic é um SDK de software livre completo para desenvolvimento de aplicativos móveis híbridos. A versão original foi lançada em 2013 e construída sobre o AngularJS e o Apache Cordova. Os lançamentos mais recentes, conhecidos como Ionic 4 ou simplesmente "Ionic", são construídos em Angular. O Ionic fornece ferramentas e serviços para o desenvolvimento de aplicativos móveis híbridos usando tecnologias da Web, como CSS, HTML5 e Sass. Os aplicativos podem ser criados com essas tecnologias da Web

e, em seguida, distribuídos por meio de lojas de aplicativos nativas para serem instalados em dispositivos, aproveitando o poder do Cordova.

O Apache Cordova é uma estrutura de desenvolvimento móvel de código aberto. Ele permite que você use tecnologias padrão da web - HTML5, CSS3 e JavaScript para desenvolvimento em várias plataformas. Os aplicativos são executados em wrappers destinados a cada plataforma e dependem de ligações de API compatíveis com os padrões para acessar os recursos de cada dispositivo, como sensores, dados, status da rede etc.

É recomendado que seja criada a Diretriz de Interface Mobile, esse diretriz tem como objetivo definir como será a interface para todos os aplicativos móveis do IBAMA, garantindo a mesma identidade visual do órgão e criando sua identidade visual para seus aplicativos.

2.3.2 Camada VIEW - WEB

Para auxiliar seu desenvolvimento e aumentar a produtividade deverão ser utilizadas as bibliotecas jQuery e AngularJS como componente principal para mediação das aplicações WEB com o designer responsível. O AngularJS para controle de interação Model Presenter.

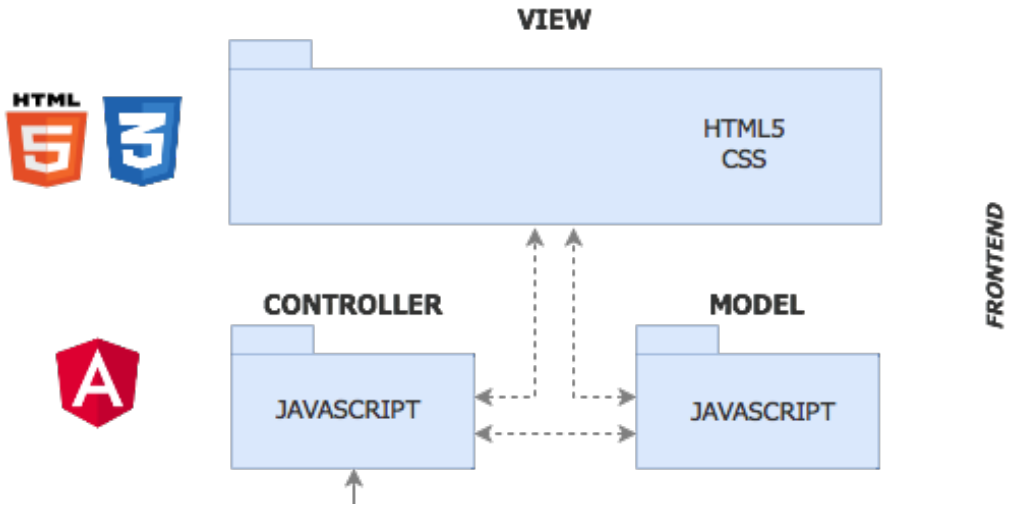

***Figura 7** - Camada Apresentação WEB*

O *Bootstrap* trata-se de links de CSS com algumas funções de JavaScript embarcadas que permitem fazer layouts completos utilizando componentes atuais. O objetivo principal e lógico do *Bootstrap* no projeto é consumir o menor tempo possível no desenvolvimento de layouts, seja uma página simples estática ou um grande portal dinâmico.

Para os sistemas do IBAMA deverá seguir a Diretriz de Interface WEB que define o padrão de componentes visuais que devem ser utilizados, além disso essa Diretriz está em conformidade com o que se pede na e-MAG (documento que define os padrões de acessibilidade do Governo Federal).

A camada de **Controller** e **Model** são utilizadas para fazer o roteamento e acesso as APIs de serviço REST que estão disponibilizadas como *microserviços* controlados pelo API Gateway, as próximas camadas não possuem retrabalho pois são independentes de front-end seja ele mobile ou WEB.

2.4 Camada API Gateway

O API Gateway está sendo provido pelo Netflix Zuul que tem como função controlar os acessos as APIs REST disponíveis pelo IBAMA, independente de tecnologias, o gateway tem um componente do SCA (Sistema de controle de acesso) do IBAMA para garantir a segurança de acesso aos dados disponibilizados pelas APIs.

O Zuul é um serviço de gateway que faz proxy de solicitações para as APIs. Ele fornece uma "porta de entrada" unificada para aplicações em geral, que permite que um navegador, aplicativo móvel ou outra interface de usuário consuma serviços de vários *hosts*. É possível utilizar o Zuul a outros projetos da Netflix, como o Eureka, para descoberta de serviços, ou usá-lo para gerenciar regras de roteamento, filtros e balanceamento de carga em todo o ecossistema.

2.5 Camada Backend (SpringBoot)

O projeto SpringBoot é uma tecnologia desenvolvido pela Spring que visa facilitar configurações de forma segura e robusta, principalmente quando se trata de microserviços.

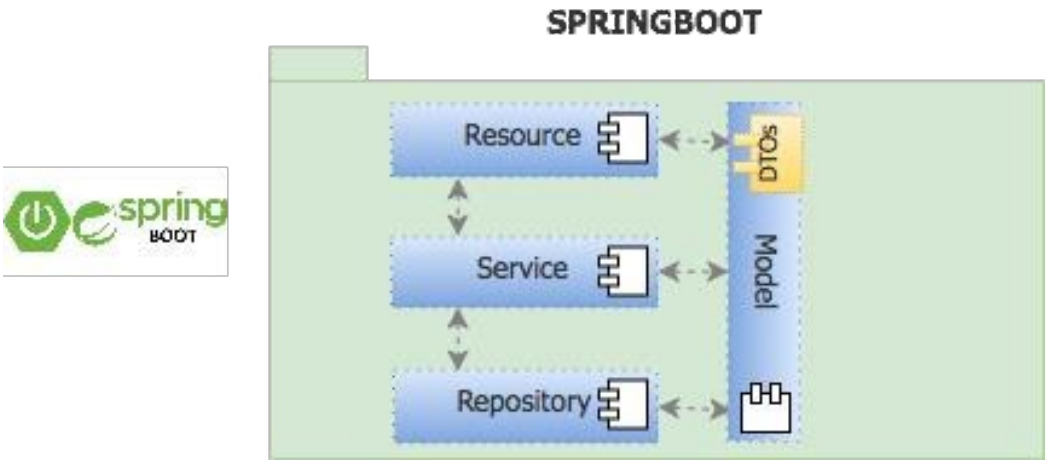

Figura 8 - Camadas SpringBoot

O SpringBoot facilita bastante o desenvolvimento por utilizar várias tecnologias já consolidadas.

2.5.1 Camada Resouce

Fachada de atendimento de todas as solicitações dos diversos aplicativos clientes esta camada responsável por expor os serviços com os recursos e operações necessárias, nesta camada que é a definição de quais são as operações, *endpoints* e métodos HTTP serão atendidos (GET, POST, DELETE, PUT, ...).

2.5.2 Camada Service

Para implementar os serviços REST deverá ser utilizada as bibliotecas da especificação Java, JAX-RS (Especificação JCP que define a API Java para serviços web RESTFul sobre protocolo HTTP). Possuiu rica estrutura de Interceptors e integração com o OAuth2 e SSO distribuído quando utilizando sobre servidores de aplicação ou frameworks como o SpringBoot.

2.5.3 Camada Repository

Camada responsável pela comunicação com fontes de dados, esta camada possuí uma grande importância e pode-se comunicar com qualquer fonte de dados (bancos de dados, unidades de *storage* ou até mesmo um outro serviço). O SpringBoot através de suas bibliotecas já disponibiliza á grande maioria de métodos utilizados para execução das operações com fontes de dados, mas também é possível implementar operações inexistentes de maneira simples e eficiente.

2.5.4 Camada Model

Camada responsável pelo mapeamento objeto relacional, mapeamento das informações do banco de dados com a aplicação.

Os DTOs são objetos construídos para transporte de informações entre as camadas, tem como objetivo guardar apenas as informações necessárias para transporte

entre todas as camadas do projeto, facilita o desenvolvimento e diminui o acoplamento de objetos.

3. Servidor de Aplicação

Dentro das definições da arquitetura é necessário a utilização de um servidor WEB, o servidor escolhido foi o APACHE 2.4 ou superior, este servidor terá como responsabilidade disponibilizar a aplicação de frontend na WEB, também a arquitetura tem como premissa que todas as aplicações WEB do IBAMA deveram fazer uso do SCA (Sistema de Controle de Acessos) e SSO (Single sing-on) quando não forem aplicativos, no caso de aplicativo será feito o uso do SCA via requisição de *tokens* utilizando preferencialmente o protocolo OAuth2.

Como o backend trata de microserviços adotou a utilização de APIs com servidores de aplicação autocontido, ou seja, a aplicação do backend não precisa de servidores de aplicação a própria aplicação já possui um servidor contido no projeto, diminuindo as dependências externas e a possibilidade de escala e disponibilidade.

4. Navegadores

Todos os projetos WEB entregues devem ser homologados nos navegadores mais utilizados, após a homologação no documento de checklist para produção deve possuir a versão do navegador que foi homologada.

Mozilla Firefox ❼ é um navegador livre e multi-plataforma desenvolvido pela Mozilla Foundation com ajuda de centenas de colaboradores.

Google Chrome ❼ é um navegador de internet, desenvolvido pela companhia Google com visual minimalista e compilado com base em componentes de código licenciado como o motor de renderização o WebKit.

Internet Explorer ou Edge ➐ Internet Explorer é uma série de navegadores web gráficos desenvolvidos pela Microsoft e inclusos como parte da linha de sistemas operacionais Microsoft Windows.

5. Sistemas Operacionais Móveis

Todos os projetos de aplicativos mobile entregues devem ser homologados nos sistemas operacionais mais utilizados, após a homologação no documento de checklist para produção deve possuir a versão da plataforma que foi homologada.

Android ➐ é um sistema operacional baseado no núcleo Linux e atualmente desenvolvido pela empresa de tecnologia Google.

iOS ➐ é um sistema operacional móvel da Apple Inc. desenvolvido originalmente para o iPhone, também é usado em iPod *touch* e iPad. A Apple não permite que o iOS seja executado em hardware de terceiros.

Windows Phone ➐ Windows Phone foi um sistema operacional para smartphones, desenvolvido pela Microsoft, que é focado no mercado consumidor, em vez do mercado empresarial.

6. Metas e Restrições da Arquitetura

Para a proposta da arquitetura, foram considerados fatores como tecnologias *open-source*, padrões recomendados e adotados pelo Governo Federal, finalidade do sistema, tipo de usuários e ambiente de execução, possibilitando a construção de diversas soluções altamente escaláveis e de evolução extremamente simples. Sendo assim, a arquitetura visa atender às seguintes características:

- Modularidade;
- Manutenibilidade;
- Extensibilidade;

- Reusabilidade; • Escalabilidade;
- Segurança.

7. Qualidade

Além dos padrões de software e arquitetural, descritos anteriormente nesta proposta de arquitetura, os seguintes elementos de qualidade de software poderão ser considerados no desenvolvimento de projetos: Novos e evolutivos.

O framework permitirá que os softwares desenvolvidos sofram análise estática de qualidade de código-fonte, através da utilização do Sonar disponível no link, http://www.sonarqube.org, o que cobrirá 7 (sete) pontos de qualidade: Arquitetura e Design, Comentários, Duplicações, Padrão de código-fonte (regras de codificação, como formatação), Unidades de Testes, Potenciais Bugs e Complexidade de código, como mostra a figura 8.

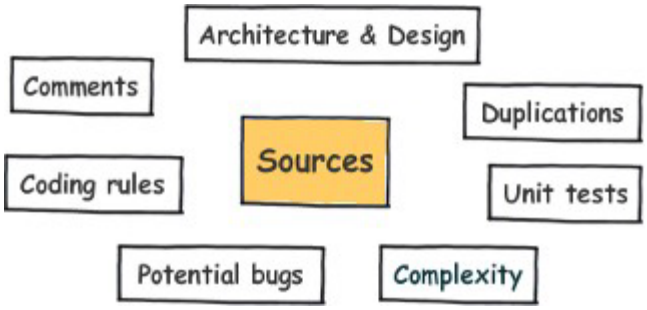

Figura 9 - *Cobertura de Qualidade - Sonar*

Desta forma, poderão ser gerados relatórios de qualidade de código a todo o momento.

Atualmente o SonarQube possuí mais de 455 regras pra Java (https://rules.sonarsource.com/java), sendo atualizado conforme novas regras.

Nas figuras 10 e 11, é possível observar um relatório geral de vários projetos controlados pela ferramenta e um relatório de um projeto específico.

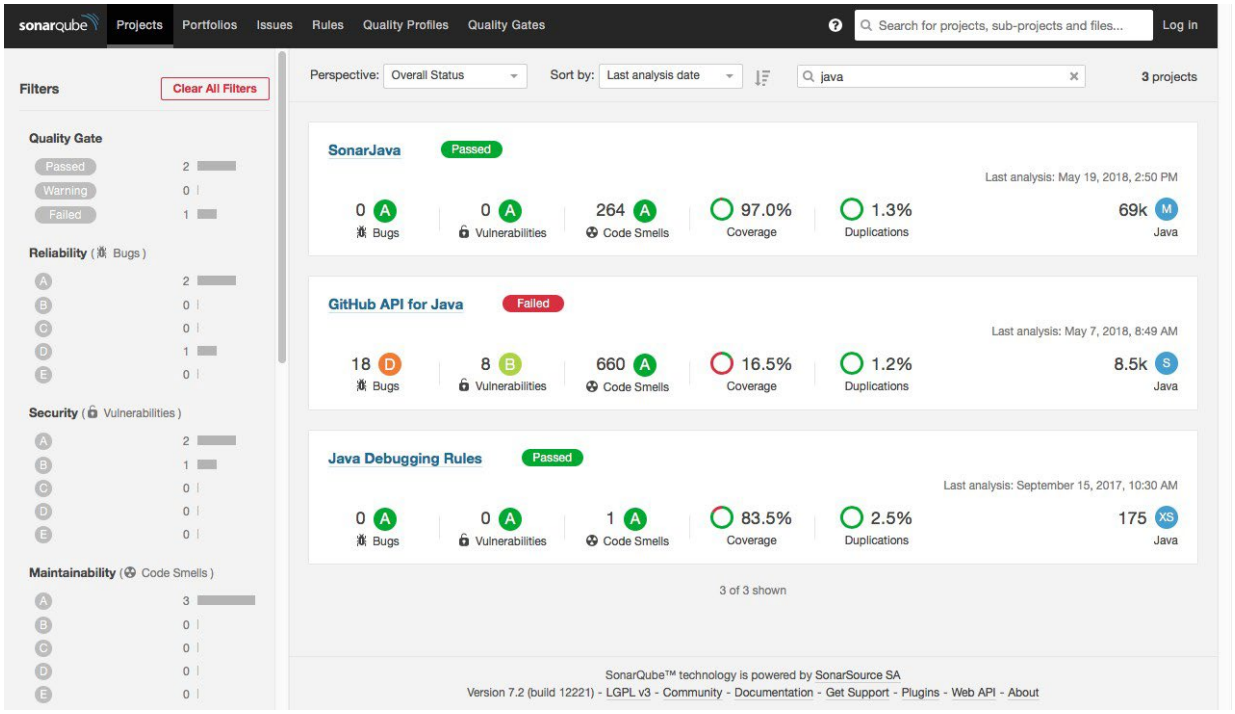

Figura 10 *- Exemplo de Dashboard SonarQube*

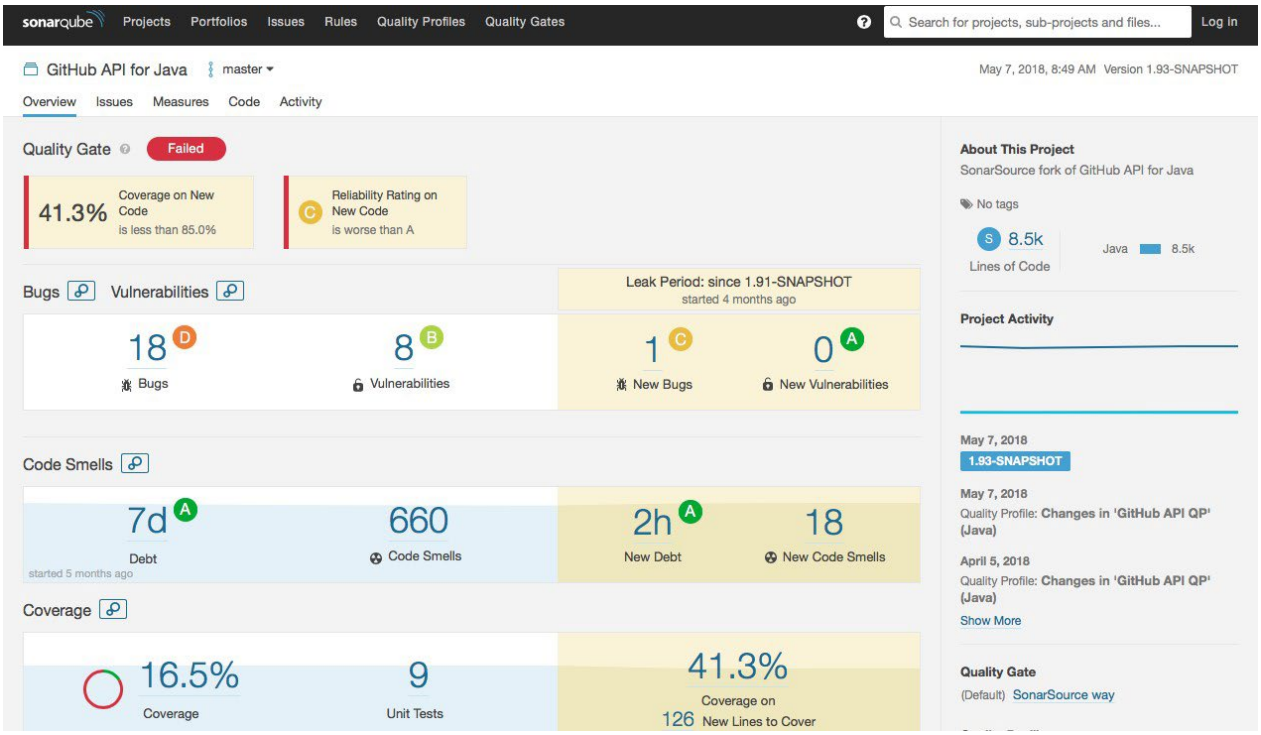

Figura 11 *- Exemplo de Dashboard detalhado de um projeto*

Todos os projetos poderão ser integrados e verificados, conforme citado anteriormente, utilizando a plataforma *Jenkins*, disponível no link: http://jenkins-ci.org. Desta forma em qualquer interação com o repositório de códigos-fonte, as alterações poderão ser analisadas e o relatório de qualidade será atualizado automaticamente, fornecendo ferramenta de acompanhamento do nível de qualidade corrente permitindo a atuação de maneira eficaz na sua melhora. O Jenkins também deverá ser utilizado dentro do conceito DevOps para geração e implantação dos pacotes em desenvolvimento, homologação e produção diminuindo divergências de versões em ambientes e possibilitando um retorno à uma versão anterior conforme necessidade.

Na figura 11 é possível observar como devemos utilizar o repositório de códigos fontes. É imprescindível que versionamento seja feito em um repositório único, garantindo

que as versões não sejam diferentes e automatizando o processo de verificação, homologação e implantação dos sistemas em produção. Além disso o repositório oficial considerado sempre será o do IBAMA (https://git.ibama.gov.br).

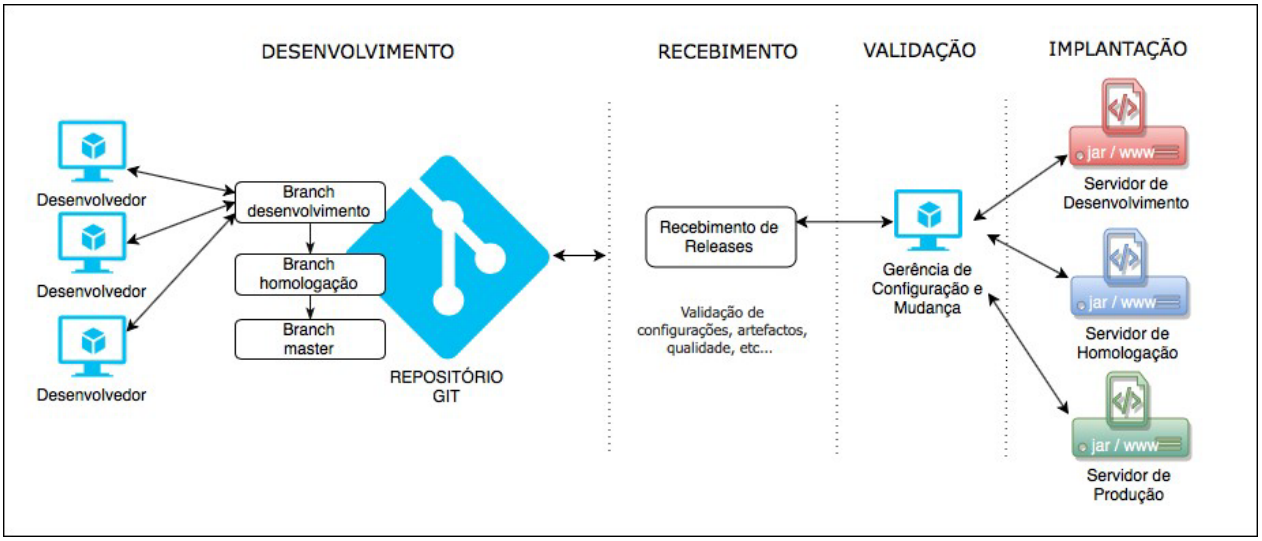

Figura 12 *- Repositório de Trabalho*

Poderá ser criado ambiente similar ao de produção que será utilizado durante o desenvolvimento. Este ambiente, chamado de ***Staging***, deverá conter versões atuais dos sistemas em desenvolvimento. O Sistema ***Jenkins*** poderá efetuar a construção e instalação dos sistemas neste ambiente diariamente ou sobre demanda, ou seja, será possível solicitar ao ***Jenkins*** que efetue a construção e instalação de qualquer sistema a qualquer momento, agindo pro-ativamente na verificação de correção de irregularidades ou melhorias.

O Processo de desenvolvimento prevê a criação de testes de unidade e integração. Para implementação dos testes de unidade poderão ser utilizados os seguintes frameworks: Junit 4, DbUnit, CDI-Unit.

Junit, disponível no link: http://junit.org, é um framework simples para codificação de testes repetíveis.

DbUnit, disponível no link: http://dbunit.sourceforge.net, é uma extensão do Junit voltada para desenvolvimento de testes orientados a banco de dados. Com ele é possível criar cenários de estado de banco de dados para verificar a manipulação destes dados no sistema em teste, evitando diversos problemas como base corrompida.

CDI-Unit, disponível no link: http://jglue.org/cdi-unit, é um executor de testes desenvolvido sobre o Junit 4. CDI-Unit é rápido, simples e isolados. Todas as configurações para execução dos testes poderão ser feitas utilizando *Annotations* e os testes serão então executados fora do container web.

A análise de cobertura de códigos-fonte pelos testes poderá ser feita pelo Sonar, gerando relatório de cobertura, mas também poderá ser feita durante o desenvolvimento. Para isso deverá ser utilizado o framework da EclEmma, o JaCoCo, disponível no link: http://www.eclemma.org/jacoco/index.html. Ele pode ser instalado em diversas IDEs do mercado. A análise de cobertura direciona o desenvolvimento dos testes evitando desperdício de esforço.

8. Distribuição

Como boa prática e facilidade no acesso aos sistemas é sugerido que sejam utilizados os prefixos nos nomes dos sistemas ao disponibilizar o endereço do mesmo, onde em produção o prefixo é subtraído, conforme exemplo abaixo:

ITEM	AMBIENTE	DESCRIÇÃO
01	Desenvolvimento	sistema.**dev**.ibama.gov.br

02	Homologação	sistema.**hom**.ibama.gov.br
03	Teste	sistema.**tst**.ibama.gov.br
04	Treinamento	sistema.**tnm**.ibama.gov.br
05	Produção	sistema.ibama.gov.br

9. Ferramentas e Tecnologias

Para implementação da arquitetura de sistemas são listadas as principais ferramentas necessárias para utilização, caso seja necessário o uso de outra ferramenta, a prioridade é sempre o uso de software livre, conforme tabela abaixo:

FERRAMENTA	DESCRIÇÃO	DOWNLOAD
	IDE de desenvolvimento de sistemas, com suporte a várias tecnologias (JAVA, PHP, C, entre outras). Possui plugin para integração com GIT.	www.eclipse.org Software Livre
	IDE de desenvolvimento de sistemas, com suporte a várias tecnologias (JAVA, PHP, C, entre outras). Possui plugin para integração com o GIT e SonarQube.	www.netbeans.org Software Livre

 Oracle SQL Developer	Ferramenta de desenvolvimento, administração, modelagem e migração de dados para bancos Oracle.	http://www.oracle.com/technetwork/developer-tools/sql-developer Software Livre
	Ferramenta para versionamento de código fonte, necessário para integração e controle de versões.	http://git-scm.com/ Software Livre
	Ferramenta de automatização de Deploy, e execução de tarefas necessárias para acompanhamento da qualidade.	http://jenkins-ci.org/ Software Livre
	Ferramenta de gerenciamento de dependências necessárias para	http://maven.apache.org/ Software Livre
	funcionamento dos projetos Java.	
	Ferramenta para verificação da qualidade do código produzido.	http://www.sonarqube.org/ Software Livre

10. Documentação dos códigos fontes

A documentação dos códigos fontes é uma etapa importante para agilizar o entendimento e facilitar o uso do framework pelos desenvolvedores, para contemplar esse requisito será utiliza a API padrão do Java para documentação (Java Doc).

Os desenvolvedores usam certos estilos de comentários e *tags* Javadoc ao documentar códigos-fonte. Um bloco de comentário em Java iniciado com /** irá iniciar

um bloco de comentário Javadoc, que será incluído no HTML gerado. Uma *tag* Javadoc começa com um "@" (arroba). Na figura 13 consta as tags principais para utilização.

Tag	Descrição
@author	Nome do desenvolvedor
@deprecated	Marca o método como *deprecated*. Algumas IDEs exibirão um alerta de compilação se o método for chamado.
@exception	Documenta uma exceção lançada por um método — veja também @throws.
@param	Define um parâmetro do método. Requerido para cada parâmetro.
@return	Documenta o valor de retorno. Essa tag não deve ser usada para construtores ou métodos definidos com o tipo de retorno *void*.
@see	Documenta uma associação a outro método ou classe.
@since	Documenta quando o método foi adicionado a a classe.
@throws	Documenta uma exceção lançada por um método. É um sinônimo para a @exception introduzida no Javadoc 1.2.
@version	Exibe o número da versão de uma classe ou um método.

Figura 13 *- Tags Javadoc*

Deste modo, a utilização das tags acima permitem gerar uma documentação do código fonte em HTML para consulta posteriores, conforme figura 14.

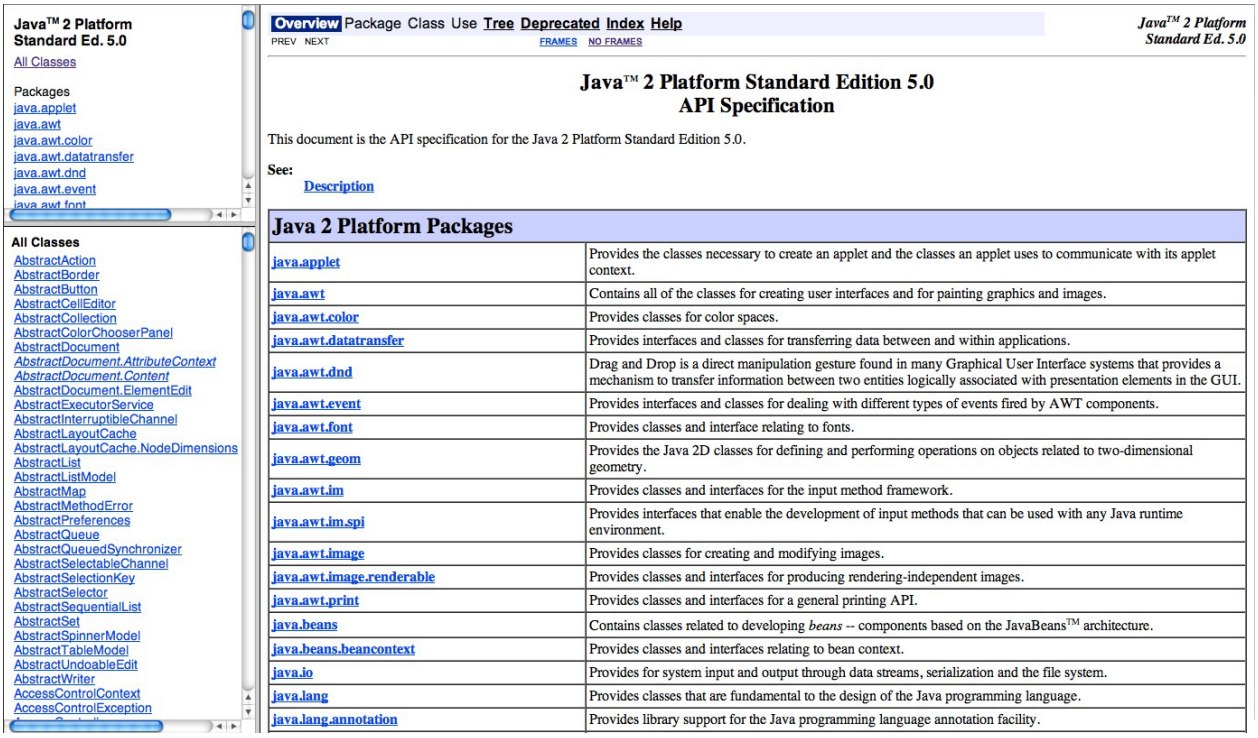

Figura 14 *- Documentação Javadoc*

11. Treinamentos

Para auxiliar no entendimento da arquitetura adotada, é interesse do IBAMA, prover treinamentos para os servidores. Dessa forma, abaixo segue detalhamento dos módulos necessários.

Módulo	Ementa	Carga Horária
Básico Programação	Desenvolvimento de competências e habilidades para utilizar a linguagem de programação C. Identificar, definir e implementar: o histórico da linguagem e sua sintaxe básica; a declaração de variáveis, os operadores, as suas bibliotecas, a execução de programas, o compilador gcc, a precisão numérica, os tipos de variáveis, a entrada e saída de dados, as estruturas de decisão, as estruturas de repetição,	24h/a
	as funções, as passagem de parâmetros, os vetores e matrizes, os arquivos, os registros, as cadeias de caracteres, as estruturas dinâmicas (ponteiros), e recursão. Implementar e analisar algoritmos em C.	
Básico WEB	Desenvolvimento de competências e habilidades na criação de aplicações em ambiente Web. Introdução à linguagem de marcação HTML, Java Script e CSS.	20h/a
Programação Java	Desenvolvimento de competências e habilidades sobre Programação Orientada o Objetos (POO) aplicados com a Linguagem Java: Conceitos de classes, objetos, herança, polimorfismo, encapsulamento, interface, atributos, métodos; O ambiente de desenvolvimento de linguagens orientadas a objetos; Os paradigmas da programação orientada a objetos, histórico da Linguagem Java; J2SE; linguagem pseudo-compilada; conceito de máquina virtual; características da Máquina Virtual Java – JVM; Fundamentos práticos; Estrutura da linguagem; Acesso a Banco de Dados; Implementação em programas dos conceitos adquiridos. Utilização de Laboratório de Linguagens para consolidação da teoria/prática.	32h/a

Java Avançado para WEB	Desenvolvimento de competências e habilidades para implementação de sistemas corporativos utilizando a especificação JEE. Utilização de Servlets, Html, Jsp, banco de dados, XML, JSTL e padrões de projetos para construção de arquitetura estável para desenvolvimento de sistemas Web. Implementação em programas dos conceitos adquiridos. Utilização de Laboratório de Linguagens para consolidação da teoria/prática.	40h/a
Trabalhando com IONIC	Desenvolvimento de uma aplicação simples e funcional utilizando IONIC e compilando em Android.	24h/a
Trabalhando com SpringBoot	Desenvolvimento de uma aplicação simples e funcional dentro do framework SpringBoot.	24h/a
Jenkins	Configuração e utilização do Jenkins.	16 h/a
	Total de Horas na formação	180 h/a

Conforme a tabela acima, os treinamentos totalizam 180 horas aulas. Essas aulas devem ser ministradas por um profissional da área com capacidades de lecionar os cursos. Todos os módulos poderão ter no máximo 20 alunos, as aulas devem ser em um laboratório equipado com Microcomputadores e projetor multimídia.

ANEXOS

- *Template* de Manual de Implantação;
- Documento de Qualidade de Código;
- Documento de *Checklist* Para Implantação em Produção;
- Vídeo de Apresentação da Arquitetura de Sistemas;
- Vídeo de Modelo de Validação de Projeto para Produção.

12. Referências

SILVEIRA, P. ; SILVEIRA G., LOPES, S., Introdução a Arquitetura e Design de Software uma visão sobre a plataforma Java, Editora: Elsivier, 1ª Edição, 2009.

FILHO, Antonio M. S., Artigo Arquitetura de Software: Desenvolvimento orientado para arquitetura – Revista 1ª Edição Engenharia de Software, link http://www.devmedia.com.br/arquitetura-de-software-desenvolvimento-orientado-paraarquitetura/8033, acessado em 24/11/2014.

Governo Federal, eMAG - Modelo de Acessibilidade em Governo Eletrônico, link: https://www.governodigital.gov.br/documentos-e-arquivos/eMAGv31.pdf, acessado em 05/05/2018.

Governo Federal, Padrões de Interoperabilidade de Governo Eletrônico, link: https://www.governodigital.gov.br/documentos-e-arquivos/ePING_v2017_20161221.pdf, acessado em 05/05/2018.

SpringBoot, Documentação oficial do produto, link: https://projects.spring.io/spring-boot/, acessado em 01/05/2018.

Netflix Zuul, Documentação oficial da comunidade, link: https://github.com/Netflix/zuul/wiki, acessado em 15/05/2018.

IONIC Framework, Documentação oficial do produto, link: https://ionicframework.com/docs/, acessado em 15/05/2018.