Diretriz Arquitetural

Instruções de preenchimento do documento

Este documento deverá seguir as seguintes considerações de preenchimento:

- **Fonte:** Arial
- **Tamanho:** 10
- **Cor:** Preta
- **Sigla e nome do Projeto:** Caixa Alta
- **[...]:** Retirar os *colchetes* ao transcrever o texto
- Quando algum item não for aplicado a este documento informar a sigla **N/A**

Diretriz Arquitetural

Gestor do Projeto
Klaymer Paz
klaymer.paz@saude.gov.br
3315-2212

Histórico de Revisão

Data	Versão	Nº Ordem de Serviço	Descrição	Autor
26/12/2018	1.0	04.2018	Criação do documento	Eduardo Nascimento

Sumário

1. INTRODUÇÃO

A proposta deste documento é definir a arquitetura básica e os conceitos fundamentais dos sistemas desenvolvidos pelo DATASUS. Estes padrões e diretrizes são apresentados em forma de visões arquiteturais que visam cobrir os principais aspectos técnicos relativos a estrutura e ao desenvolvimento dos sistemas. Segundo Buschman et ali. (1996), um padrão arquitetural expressa um esquema de organização estrutural fundamental para sistemas de software, fornece um conjunto de subsistemas predefinidos, especifica suas responsabilidades e inclui regras e diretrizes para organizar as relações entre eles.

A Arquitetura de Referência gera um aumento na Eficiência de Entrega de Projetos de Novas soluções tecnológicas, pois:

A Arquitetura de Referência em um contexto de T.I. Integradora é um mapa que garante as entregas de projetos de T.I. de maneira coordenada e aderente à estratégia corporativa.

A Arquitetura de Referência é, além de um mapa, um manual de como configurar cada componente que será integrado em uma nova solução tecnológica. Desta maneira, a quantidade de retrabalho e o tempo de execução na reconfiguração de componentes serão consideravelmente reduzidos e isso causa um aumento da eficiência nas entregas de Projetos de TI.

O documento necessita de revisões periódicas. Normalmente, novas necessidades de negócios motivam revisões na Arquitetura de Referência para que ela possa atender a essas necessidades. O Grau de alinhamento desta Arquitetura com a estratégia da empresa pode fornecer um caminho de duas vias, ou seja, inovações tecnológicas podem gerar novas capacidades ao modelo de negócios da empresa.

Para Booch et ali. (2005), a arquitetura é o conjunto de decisões significativas acerca dos seguintes itens:

- A organização do sistema de software;
- A seleção dos elementos estruturais e suas interfaces, que compõem o sistema;
- Seu comportamento, conforme especificado nas colaborações entre esses elementos;

- A composição desses elementos estruturais e comportamentais em subsistemas progressivamente maiores.
- O estilo de arquitetura que orienta a organização: os elementos estáticos e dinâmicos e suas respectivas interfaces, colaborações e composição.

Devido às características de manutenção dos sistemas legados, este documento não aborda ou define a sua estrutura, pois estes não são cobertos por esta proposta arquitetural.

1.2 Representação Arquitetural

Seguindo o modelo 4+1 de representação arquitetural definido por Philippe Kruchten, este documento cobrirá as seguintes visões arquiteturais:

- **Visão Lógica:** Abrange as classes, interfaces e colaborações que formam o vocabulário do problema e de sua solução;
- **Visão de Implementação:** Abrange os componentes e os artefatos utilizados para a montagem e fornecimento do sistema sico;
- **Visão do Processo:** Mostra o fluxo de controle entre as várias partes, incluindo mecanismos de concorrência e sincronização;
- **Visão de Implantação:** Abrange os nós que formam a topologia de hardware em que o sistema é executado.

Figura 1: Modelo 4+1

2. REQUISITOS E RESTRIÇÕES ARQUITETURAIS

Os requisitos e restrições arquiteturais são definições realizadas em consenso entre a área de arquitetura e a gerência do projeto, influenciadas pelas necessidades dos usuários dos sistemas. Estas

definições influenciam diretamente no planejamento da arquitetura e podem ter diferentes origens, como por exemplo, requisitos de segurança, portabilidade, confiabilidade e outros.

2.1 Restrições

As seguintes restrições de requisito e de sistema possuem uma relação significativa com a arquitetura:

- Os sistemas poderão ser acessados por meio de rede local ou Internet:
- A linguagem de desenvolvimento utilizada para os sistemas devem seguir o documento de referência especificado e definido da linguagem a ser utilizada. A linguagem será embasada nas características do sistema e recursos necessários;
- O Servidor de Aplicações para os sistemas desenvolvidos devem seguir o Guia de referência Especificado para cada linguagem.
- O Sistema Operacional que dará suporte aos serviços da aplicação deverá ser preferencialmente software livre ou, em casos específicos e justificados, aquele definido segundo arquitetura de referência da linguagem do serviço.
- Os sistemas serão instalados em ambiente clusterizado;
- Os sistemas deverão ser acessíveis para deficientes visuais;
- Os sistemas deverão utilizar o SCPA para autenticação e autorização
- Sendo um sistema Web, ele deverá funcionar com os navegadores Internet Explorer 8 ou superior, Firefox 20 ou superior e Chrome 24 ou superior.

2.2 Requisitos e Soluções

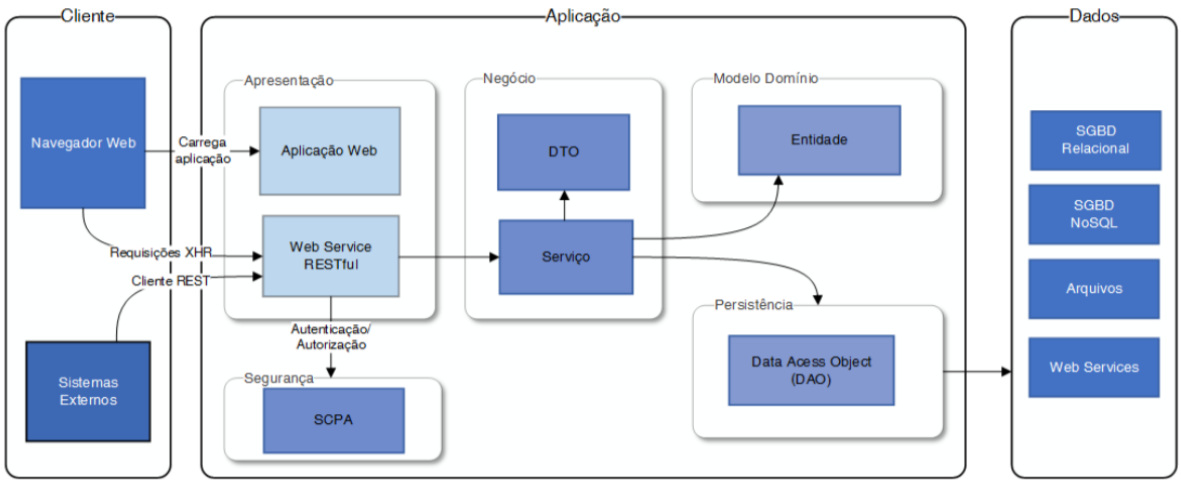
Requisito	Solução Arquitetural
Portabilidade de SGBDs	Deverá utilizado um framework de mapeamento objeto- relacional (ORM) na camada de persistência para a abstração do acesso ao banco de dados. Através do uso de um framework é possível a portabilidade entre fornecedores de bancos de dados sem impacto significativo na aplicação.
Reuso de componentes	Todos os componentes construídos deverão ser autorizados e projetados em conjunto com a equipe de arquitetura, para garantir um alto grau de encapsulamento de forma a permitir o seu reuso em outros projetos.
Utilização do padrão MVC	Definir a arquitetura dos sistemas em camadas, atribuindo responsabilidades a cada uma delas.

Controle de inatividade	Entende-se por inatividade o período em que o usuário do sistema não executa operações que gerem a comunicação com o servidor de aplicação. Complementando esse mecanismo, será exibido na interface do usuário um relógio com tempo decrescente. Quando o relógio indicar que acabou o tempo, será acionada automaticamente a funcionalidade de logoff do sistema.
Interfaces de usuário amigáveis e dinâmicas	A interface de usuário será desenvolvida considerando a usabilidade e acessibilidade da aplicação.
Autenticação e Autorização	A autenticação da camada de serviço REST será feita pelo framework Oauth 2.0 em conjunto com o Sistema de Cadastro e Permissão de Acesso do Ministério da Saúde – SCPA.
Envio de e-mail	Será utilizada a estrutura de servidores SMTP do DATASUS para o envio de e-mail.
Paginação de dados	A recuperação de uma grande quantidade de dados para apresentação ao usuário do sistema provoca o consumo desnecessário de memória e processamento. Será adotada a estratégia de paginação sob demanda, realizada com o uso dos recursos de banco de dados e do framework ORM utilizado. A quantidade máxima de objetos que pode compor uma página é de 50. O usuário poderá escolher entre 10, 20, 30, 40 e 50 objetos por página.
Processamento em lote	Rotinas de processamento em lote devem ser implementadas em módulos independentes nos projetos e sua execução deverá suportar agendamento.
Exclusão logica de dados	Por restrições legais, informações não serão excluídas do banco de dados e em função disso a exclusão logica de dados será aplicada. Cada tabela possuirá um atributo (ST_REGISTRO_ATIVO) que indicará se o registro está ou não ativo. Caso exista um mecanismo de registro de trilhas de auditoria na tabela, a exclusão logica é opcional.
Trilhas de auditoria	O registro de trilhas de auditoria será feito automaticamente através de mecanismo de triggers no banco de dados. Fica na responsabilidade da aplicação, através de um framework comum do DATASUS, definir os valores dos atributos que serão gravados na trilha de auditoria. O framework do DATASUS irá prover uma implementação padrão que irá recuperar os dados e endereço IP do usuário e atribuir estes dados à sessão do banco de dados a cada operação de inclusão, alteração e exclusão de registros.
Geração de relatórios	Os relatórios serão gerados com o framework JasperReports. O design será feito com o uso do IDE Ireport.
Armazenamento de arquivos	O armazenamento de arquivos nas aplicações poderá ser feito em sistema de arquivos ou banco de dados. A estratégia de armazenamento deverá ser analisada de acordo com o volume de dados, características dos arquivos armazena- dos e ciclo de vida dos arquivos no sistema. No armazenamento em sistema de arquivos, será criada uma pasta específica por aplicação em um servidor de arquivos configurada com um usuário específico. Políticas de expurgo e backup deverão ser definidas. Na estratégia de armazenamento no banco de dados, os arquivos deverão ser armazenados em uma tabela específica na aplicação e sua referência (FK) será armazenada na tabela que utilizar este arquivo.

Aderência ao E-mag 3.0	Deverão ser seguidas as boas práticas para o desenvolvimento de aplicações/sites/portais contidas na documentação do E-mag 3.0.
Segurança dos Sistemas	Os sistemas deverão implementar as soluções de segurança para endereçar os dez maiores riscos em aplicações Web publicados pela comunidade OWASP. A implementação dessas soluções será baseada no Guia de Segurança de Software do DATASUS.

3. VISÃO LÓGICA

3.1 Visão Geral da arquitetura



3.2 Camada de Apresentação

A camada de apresentação é responsável por disponibilizar aos atores do sistema a possibilidade de interação com os objetos gerenciados pela aplicação. Esta camada é principalmente representada na forma de interfaces gráficas para os usuários ou serviços disponibilizados a outras aplicações por protocolos de comunicação.

As principais atividades atribuídas a camada de apresentação são:

- Exibir informações para usuário, tanto em telas interativas, quanto de forma estática, através de relatórios em tela ou em formato de arquivo texto;

- Fornecer uma estrutura de Helpers (componentes auxiliares que eliminam o trabalho repetitivo e despadronizado, encapsulando funcionalidades comuns em funções, métodos ou procedimentos);
- Disponibilizar protocolos de comunicação pré-definidos e acordados com outras aplicações.

Em nível de implementação, esta camada será dividida em 2 subcamadas, a saber:

Interface Gráfica Web - Front-end: Esta subcamada terá a responsabilidade de realizar a interação com o usuário do sistema, principalmente na forma de formulários e relatórios. Por serem processadas e visualizadas no navegador do usuário, as interfaces serão compostas basicamente por elementos HTML, imagens, fontes e código JavaScript.

Web Service RESTful - Back-end: Esta subcamada terá a responsabilidade de disponibilizar os recursos para a interface gráfica e para outros sistemas que irão consumir os dados da aplicação, através de uma camada de serviços Web utilizando o padrão arquitetural REST.

A estrutura desta camada, bem como seus componentes principais, está representada no diagrama abaixo:

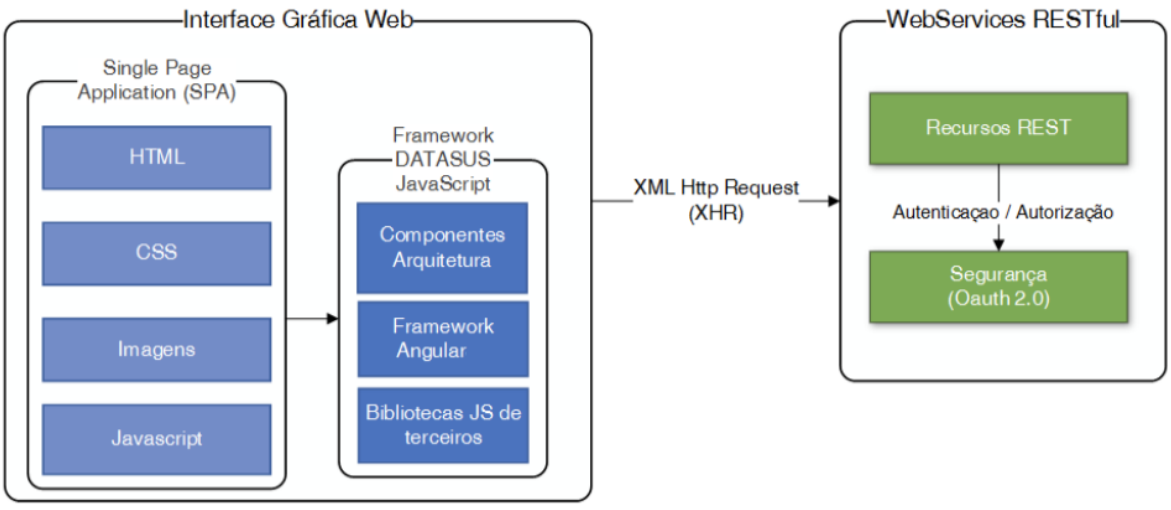


Figura 2: Modelos e camadas

3.2.1 Interface gráfica Web - Front-end

A Interface das aplicações será desenvolvida baseada no conceito de Single Page Application (SPA), onde os recursos utilizados na interface, composta de telas, recursos gráficos e scripts, são carregados uma única vez no navegador Web do usuário da aplicação.

A partir do carregamento inicial, serão feitas apenas operação de envio e recuperação de dados através de requisições baseadas em XMLHttpRequest (XHR) assíncronas a partir do navegador Web. Os objetos que irão trafegar entre o cliente e servidor serão serializados no formato JSON (Javascript Object Notation).

3.2.2 WebService RESTful - Back-end

Esta camada de serviço consistirá em Webservices que irão expor os recursos da aplicação através de uma API REST.

A arquitetura REST (Representational State Transfer ou Transferência do Estado Representacional) tem como principal característica a implementação totalmente baseada no protocolo HTTP. Cada requisição REST contém toda informação necessária para a indicar a ação e dados envolvidos na operação efetuada. Basicamente as requisições HTTP serão compostas de um cabeçalho HTTP e um corpo de mensagem com os dados. Os recursos serão acessados através de URIs(Uniform Resource Identifier), strings que descrevem a localização de um recurso HTTP.

Um ponto importante que deve ficar claro, é que na arquitetura REST a exposição dos recursos para a camada de apresentação será feita através de URIs apontando para a localização do recurso e a operação desejada será definida pelo método de requisição HTTP. Este tipo de implementação difere de chamadas remotas onde o foco é no método (operação) executado e este método é informado na própria URL de requisição.

3.2.3 Métodos HTTP

Nas requisições serão utilizados os métodos de requisição HTTP (GET, POST, PUT, DELETE) para executar a ação apropriada.

Método GET

Utilizado para recuperar determinado recurso. Exemplo: http://url.sistema/rest/usuarios/12

Recupera o usuário de código 12.

Método POST

Utilizado para incluir um registo em determinada coleção ou processar determinada operação em uma coleção ou recursos. Exemplos:

http://url.sistema/rest/usuarios

Body:

```
{
        'user': 'nomeUsuario',
        'cpf': 99999999999,
        ...
}
```

Nesse exemplo, no corpo da mensagem, seria enviado o objeto que seria incluído na coleção de usuários.

http://url.sistema/rest/usuario/relatorios/indicadores

Nesse exemplo seria gerado um relatório de indicadores de usuários.

Método PUT

Incluir ou alterar um recurso específico. Exemplo:

http://url.sistema/rest/usuarios/13

Body:

```
{
        'user': 'novoNomeUsuario',
        'cpf': 99999999999,
        …
}
```

Nesse exemplo no corpo da mensagem seria enviado o objeto que seria incluído na coleção de usuários e ele seria incluído com a identificação 13.

Método DELETE

Excluir um recurso específico. Exemplo:

http://url.sistema/rest/usuarios/13

Nesse exemplo seria para excluir o usuário com o identificador de valor 13.

Para nomenclaturas, a recomendação é de se usar '-'. Dessa forma, a leitura fica clara e mais fácil para identificação. Exemplo:

http://url.sistema/rest/usuario/relatorios/lista-indicadores-gerenciais

Logo, **lista-indicadores-gerenciais**, fica claro o endpoint que está sendo utilizado.

3.2.4 Status HTTP

Os resultados irão retornar em objetos padrão (envelopes) no formato definido pelo DATASUS e deverão possuir o código de status HTTP apropriado. Os seguintes códigos de status HTTP deverão ser utilizados nos retornos da aplicação

200 OK: operação foi efetuada com sucesso

201 Created: recurso criado com sucesso

202 Accepted: utilizado para indicar o sucesso no início de operações assíncronas

400 Bad Request: problemas na requisição

401 Unauthorized: acesso não autorizado

403 Forbidden: acesso negado

404 Not Found: recurso não encontrado

500 Internal Server Error: erro em tempo de execução

3.2.5 Serialização dos dados entre camadas

Por padrão, os dados do corpo das mensagens que irão trafegar entre a camada REST e a camada de apresentação Web, deverão estar serializados no formato JSON (Javascript Object Notation).

3.3 Autenticação/Autorização

A autenticação e autorização de acesso aos recursos da API REST da aplicação será feito utilizando o protocolo Oauth 2.0 em conjunto com o Sistema de Cadastro e Permissão de Acesso do Ministério da Saúde - SCPA.

O OAuth 2 é uma estrutura de autorização que permite que os aplicativos obtenham acesso limitado às contas de usuários em um serviço. Ele funciona delegando a autenticação de usuário ao serviço que hospeda a conta do usuário, e autorizando aplicações de terceiros a acessar a conta do usuário. O OAuth 2 fornece fluxo de autorização para aplicações web e desktop, e para dispositivos móveis.

As funcionalidades básicas de autenticação, autorização e recuperação de permissões de acesso serão disponibilizadas pelo framework de desenvolvimento do DATASUS.

3.3.1 Processo de Autenticação/Autorização

O processo de autenticação/autorização ocorre no consumo dos recursos REST pela camada Web da aplicação. Será utilizado o protocolo OAuth 2.0 em conjunto com o Sistema de Controle de Permissão e Acesso (SCPA) do DATASUS.

3.3.2 Processo de Integração com SCPA

A integração com o SCPA pode ser feita através do consumo do WebService SOAP e através do authorization server utilizando o protocolo Oauth2. A implementação do cliente do WebService será provida através de uma biblioteca com o componente de acesso e uma implementação referência no framework do DATASUS.

Para realizar as chamadas ao sistema SCPA (web service) é necessário que seja informado o e-mail, senha e sigla do projeto, a senha deverá ser cifrada pelo cifrador disponibilizado pelo DATASUS ("cifrador-scpa").

A autenticação e aquisição das permissões do usuário ao sistema é realizado em dois momentos, de acordo com as especificações do sistema SCPA:

O sistema SCPA autentica o usuário e retorna as permissões disponíveis para o usuário de acordo com o sistema; e

É realizada uma segunda conexão com o SCPA para a busca de permissões de acordo com o perfil retornado no passo um.

O framework DATASUS de desenvolvimento implementa esse processo de autenticação e recuperação de permissões de acesso.

3.3.3 Papéis Oauth

O OAuth define quatro papéis:

- Proprietário do Recurso
- Cliente
- Servidor de Recurso
- Servidor de Autorização

O proprietário do recurso é o usuário que autoriza uma aplicação a acessar sua conta. O acesso da aplicação a conta do usuário é limitado ao "escopo" da autorização concedida (por exemplo acesso para leitura ou escrita).

O servidor de recurso hospeda as contas de usuário protegidas, e o servidor de autorização verifica a identidade do usuário e então emite tokens de acesso para a aplicação.

O cliente é a aplicação que quer acessar a conta do usuário. Antes de fazer isso, ela deve ser autorizada pelo usuário, e a autorização deve ser validada pela API.

O fluxo, de forma detalhada, é:

A aplicação solicita autorização para acessar recursos do serviço do usuário. Se o usuário autorizar a solicitação, a aplicação recebe uma concessão de autorização. A aplicação solicita um token de acesso ao servidor de autorização (API) através da autenticação de sua própria identidade, e da concessão de autorização. Se a identidade da aplicação está autenticada e a concessão de autorização for válida, o servidor de autorização (API) emite um token de acesso para a aplicação. A autorização está completa. A aplicação solicita o recurso ao servidor de recursos (API) e apresenta o token de acesso para autenticação se o token de acesso é válido, o servidor de recurso (API) fornece o recurso para a aplicação.

3.4 Camada de Aplicação

O modelo da aplicação é um conjunto de componentes que formam o núcleo do sistema. Estes componentes são compostos por objetos de domínio, serviços de negócio, persistência, componentes de integração e classes auxiliares.

3.4.1 Domínio

Contém as entidades que representam os objetos de domínio da aplicação. Em geral, as entidades de domínio serão objetos mapeados a tabelas de um banco de dados relacional utilizando um framework de ORM (Object Relational Mapping), porém isso não é uma regra.

3.4.2 Negócio

A camada de negócio é composta por uma fachada de serviços de negócio (Service Facade), componentes de integração (clientes, mensageria), objetos de transferência de dados (DTOs), entre outros.

O controle transacional da aplicação é feito nesta camada, especificamente na fachada de serviços de negócio.

É opcional o uso de DTOs nas aplicações, porém é recomendado o seu uso nos seguintes casos:

- Quando é necessária uma representação simplificada de uma entidade com uma árvore de propriedades extensa, com o intuito de reduzir o volume de dados trafegados para a camada de apresentação.
- Criar uma independência da camada de negócio da camada de persistência;
- Criar objetos com propriedades e valores oriundos de entidades distintas;

No caso da utilização de DTOs, é recomendado que seja utilizado um framework de mapeamento para a conversão entre entidades e DTOs.

3.4.3 Persistência

A camada de persistência é responsável por abstrair o acesso aos dados da aplicação e prover uma implementação da infraestrutura básica de acessos às fontes de dados para as operações de criação, recuperação, atualização e exclusão (CRUD).

Na implementação da camada de persistência, a utilização de um framework ORM e do padrão DAO (Data access object ou Objeto de acesso a dados), simplificam a tarefa de construção, permitindo uma melhor produtividade e padronização nas questões de persistência.

Os frameworks serão especificados para a linguagem apropriada no documento de referência da linguagem a ser utilizada.

4. VISÃO DE PROCESSOS

4.1 Segurança

4.1.1 OWASP

Os sistemas desenvolvidos no DATASUS deverão utilizar as melhores práticas em relação a segurança de software, baseadas na lista dos dez maiores riscos definidos pela OWASP (The Open Web Application Security Project), comunidade aberta, dedicada a capacitar as organizações a desenvolver, adquirir e manter aplicações confiáveis.

De acordo com a OWASP, os dez maiores riscos são:

A1 - Injeção de Código:

As falhas de Injeção, tais como injeção de SQL, de SO (Sistema Operacional) e de LDAP, ocorrem quando dados não confiáveis são enviados para um interpretador como parte de um comando ou consulta. Os dados manipulados pelo atacante podem iludir o interpretador para que este execute comandos indesejados ou permita o acesso a dados não autorizados.

A2 - Quebra de autenticação e Gerenciamento de sessão:

As funções da aplicação relacionadas com autenticação e gerenciamento de sessão geralmente são implementadas de forma incorreta, permitindo que os atacantes comprometam senhas, chaves e tokens de sessão ou, ainda, explorem outra falha da implementação para assumir a identidade de outros usuários.

A3 - Cross-Site Scripting (XSS):

Falhas XSS ocorrem sempre que uma aplicação recebe dados não confiáveis e os envia ao navegador sem validação e filtro adequados. XSS permite aos atacantes executarem scripts no navegador da vítima que podem "sequestrar" sessões do usuário, desfigurar sites, ou redirecionar o usuário para sites maliciosos.

A4 - Referência insegura e Direta a Objetos:

Uma referência insegura e direta a um objeto ocorre quando um programador expõe uma referência a implementação interna de um objeto, como um arquivo, diretório, ou registro da base de dados. Sem a verificação do controle de acesso ou outra proteção, os atacantes podem manipular estas referências para acessar dados não autorizados.

A5 - Configuração incorreta de segurança:

Uma boa segurança exige a definição de uma configuração segura e implementada na aplicação, frameworks, servidor de aplicação, servidor web, banco de dados e plataforma. Todas essas configurações devem ser definidas, implementadas e mantidas, já que geralmente configuração padrão é insegura. Adicionalmente, o software deve ser mantido atualizado.

A6 - Exposição de Dados Sensíveis:

Muitas aplicações web não protegem devidamente os dados sensíveis, tais como cartões de crédito, IDs fiscais e credenciais de autenticação. Os atacantes podem roubar ou modificar esses dados desprotegidos com o propósito de realizar fraudes de cartões de crédito, roubo de identidade, ou outros crimes. Os dados sensíveis merecem proteção extra como criptografia no armazenamento ou em trânsito, bem como precauções especiais quando trafegadas pelo navegador.

A7 - Falha de Função para Controle de Nível de Acesso:

A maioria das aplicações web verificam os direitos de acesso em nível de função antes de tornar essa funcionalidade visível na interface do usuário. No entanto, as aplicações precisam executar as mesmas verificações de controle de acesso no servidor quando cada função é invocada. Se estas requisições não forem verificadas, os atacantes serão capazes de forjar as requisições, com o propósito de acessar a funcionalidade sem autorização adequadas.

A8 - Cross-Site Request Forgery (CSRF):

Um ataque CSRF força a vítima que possui uma sessão ativa em um navegador a enviar uma requisição HTTP forjada, incluindo o cookie da sessão da vítima e qualquer outra informação de autenticação incluída na sessão, a uma aplicação web vulnerável. Esta falha permite ao atacante forçar o navegador da vítima a criar requisições que a aplicação vulnerável aceite como requisições legítimas realizadas pela vítima.

A9 - Utilização de Componentes Vulneráveis Conhecidos:

Componentes, tais como bibliotecas, frameworks, e outros módulos de software quase sempre são executados com privilégios elevados. Se um componente vulnerável é explorado, um ataque pode causar sérias perdas de dados ou o comprometimento do servidor. As aplicações que utilizam

componentes com vulnerabilidades conhecidas podem minar as suas defesas e permitir uma gama de possíveis ataques e impactos.

A10 - Redirecionamento e Encaminhamentos Inválidos:

Aplicações web frequentemente redirecionam e encaminham usuários para outras páginas e sites, e usam dados não confiáveis para determinar as páginas de destino. Sem uma validação adequada, os atacantes podem redirecionar as vítimas para sites de phishing ou malware, ou usar encaminhamentos para acessar páginas não autorizadas.

A implementação de segurança nas aplicações deverá ser baseada no Guia de Segurança de Software publicado pelo DATASUS.

4.2 Processo de Registro de Trilhas de Auditoria

O processo de registo de trilhas de auditoria é executado por triggers no banco de dados em tabelas previamente configuradas para esta operação.

Assim que configuradas, as tabelas irão disparar triggers que irão automaticamente gerar o registro das operações de inclusão, exclusão e alteração. Porém, será necessário que as variáveis da sessão do usuário sejam informadas a sessão do banco de dados para que o registro do usuário logado e IP da máquina do usuário sejam armazenados corretamente na operação.

O framework do DATASUS já possui uma implementação base que executa essa tarefa de uma forma transparente para o desenvolvedor. Porém, será necessário que o usuário utilize o framework de forma apropriada para a execução correta dessa funcionalidade.

5. VISÃO DE IMPLEMENTAÇÃO

As visões de implementação demonstradas nesta seção são implementações de referência considerando os frameworks e componentes utilizadas em cada camada da arquitetura.

5.1 Implementação Camada de Apresentação - Interface Gráfica Web

A arquitetura e frameworks utilizados na camada de apresentação Web serão comuns a todos os sistemas. Desta forma, os serviços Web REST e camada de negócio poderão ser implementados na linguagem a ser definida pela necessidade do serviço e definida em documento de referência de cada uma das linguagens.

A implementação de referência da camada de apresentação pode ser embasada na utilização dos seguintes frameworks:

- Angular 7 (https://angular.io/): framework JavaScript que trabalha no conceito de menor manipulação possível do DOM e faz uso de padrões de projeto como MVC, MVVM e MVP.
- RequireJS (http://requirejs.org/): framework para controle de carregamento assíncrono de javascripts na conceituação de AMD (Asynchronous Module De nition) e na possibilidade de uso do CommonJS;
- Jquery (http://jquery.com/): biblioteca composta de vários componentes que facilitam a utilização e/ou manipulação do DOM.
- Twitter Boostrap (http://getbootstrap.com/): framework de utilização visual responsável pela estruturação de interfaces a nível de HTML, CSS e determinados componentes JS.

5.1.1 Componentes de referência Implementação Javascript (SPA)

A camada de apresentação de uma aplicação é dividida de maneira modular e componentizada. Dentro da APP temos a camada de componentização e configuração comum da aplicação, e os módulos que irão compor a aplicação. Essas camadas fazem uso direto da API e dos componentes da arquitetura DATASUS.

Componentes Comuns

A componentização comum a aplicação é composta de dados de configuração e dados de controle. Todo tipo de informação criada aqui é compartilhada e/ou estendida pelos módulos da aplicação.

Módulos da Aplicação

A aplicação criada pode conter N módulos. Cada módulo pode representar um caso de uso e conter suas funcionalidades. Todos os módulos da aplicação ficam armazenados no componente PAGES.

Um módulo do framework DATASUS possui uma componentização específica. A alteração dos componentes pode gerar problemas para a aplicação e não é indicado a adição ou remoção de novos componentes.

5.2 Componentes de Referência Implementação Camada de Aplicação

A implementação da Camada de Aplicação no DATASUS deve ser feita utilizando a Diretriz de referência da linguagem.

5.2.1 WebService RESTFUL

A camada de apresentação composta do WebService RESTful será implementada utilizando o framework Spring MVC em conjunto com o framework DATASUS REST WEB, contendo as implementações dos objetos DTO de resposta, rotinas de tratamento de exceções, configurações de conversão entre objetos Java e JSON e outros recursos.

Na utilização deste framework, será criada uma classe Controller com a anotação @RestController, desta forma o componente irá expor os métodos desejados em recursos REST.

5.2.2 Entidades

Entidades são objetos que possuem uma identidade de negócios distinta que é separada dos valores de seus atributos. Duas entidades são diferentes mesmo se os valores de seus atributos forem os mesmos e não podem ser utilizadas intercambiavelmente. Identificar entidades é importante porque elas correspondem a conceitos do mundo real que são centrais para o modelo de domínio do sistema.

As regras para o mapeamento de Entidades são:

Especificar o mapeamento objeto-relacional, com o uso de anotações diretamente nos atributos das entidades. Entidades que possuam uma collection como atributo @OneToMany, o atributo em questão não deve possuir um método setter e a collection já deve estar inicializada. Desta maneira a entidade estará de acordo com a boa-prática java de não existir collection nula, mas sim uma collection vazia. O motivo para não existir um método setter, é que a coleção relacionada ao objeto não deve ser substituída, mas sim exposta para ser trabalhada através de seus métodos get(), remove(), clear(), etc.

Evitar sempre que possível as chaves compostas. Buscar a solução de construção da chave substituta de auto-incremento. Se em determinado caso de uso composto por bases legadas, por exemplo, exija a chave composta, usar sempre a anotação @EmbeddedId.

Não deve ser utilizada ferramenta para a realização do mapeamento automático de entidades a partir do banco de dados.

A nomenclatura dos atributos deve ser a mais clara possível, facilitando a leitura do código. Não devem ser utilizados os nomes de campos de tabela como nomes de atributos.

Não utilizar os prefixos “TB” ou “RL” para indicar o tipo de tabela que a entidade representa.

Mapeamento de relacionamentos @ManyToMany em que a tabela associativa possua apenas chaves estrangeiras deve ser feito com a anotação @JoinTable, evitando a criação de uma nova entidade para representar apenas a tabela associativa.

Utilizar relacionamentos com FetchType.EAGER apenas em situações em que for necessário recuperar sempre os dados das entidades relacionadas. Se necessário, deverá ser utilizado apenas em mapeamentos @OneToOne ou @ManyToOne.

Objetos do tipo Byte[] devem ser criados externamente a classe e referenciados com JOIN. Dessa forma, utilizar estratégias de recuperação específicas. É conhecido que a implementação do hibernate do JPA ignora o atributo lazy em objeto binários, SEMPRE recuperando os valores, e isso pode ocasionar problemas sérios de performance nas aplicações.

5.2.3 Repositórios

Repositórios gerenciam coleções de entidades e definem métodos para encontrar e remover entidades. Um repositório encapsula o framework de persistência e é constituído por uma interface e uma classe de implementação. A interface define os métodos que podem ser chamados pelo cliente do repositório, e a classe de implementação implementa a interface chamando o framework de persistência.

5.2.4 Exceções

As regras para o tratamento de exceções:

A aplicação possui um módulo para tratamento de exceções. Dessa forma, toda exceção deve ser enviada e tratada pelo módulo de tratamento de exceções.

5.2.5 Mapper

O uso de mapper para o projeto busca facilitar o mapeamento de objeto para objeto.

O uso do Mapper é simples, bastando instanciá-lo para em seguida chamar seu método map(..) que popula o DTO e recebe o objeto fonte e o tipo do DTO.

É importante mencionar que um Object-to-Object Mapper não serve só para mapear o domínio para um DTO. Há várias outras motivações para uma hierarquia paralela. De qualquer forma um object-to-object mapper assume essa parte trabalhosa de copiar os valores entre objetos.

5.2.6 Controle Transacional

A JTA (Java Transaction API) será empregada em conjunto com os recursos do Spring Framework. Sua configuração será feita no arquivo de contexto do Spring. Caso a aplicação utilize mais de um recurso (exemplo datasource e JMS) em uma mesma transação, ambos deverão implementar a interface javax.transaction.xa.XAResource.

A demarcação de transações deverá ser feita sempre nos Services Facades com o uso da anotação @Transactional.

Atenção especial deve ser dada na configuração da transação em função de exceções. Exceptions que estendam RuntimeException provocam automaticamente a realização de rollback em uma transação, Checked exceptions não. Para provocar o rollback as checked exceptions devem ser informadas na propriedade rollbackForClassname da anotação @Transactional.

5.2.7 Processamento em lote

O termo lote ou batch é bastante comum para programadores de Mainframe. As aplicações batch podem ser consideradas como programas ou rotinas executadas em segundo plano que possuem um fluxo a percorrer e sem a intervenção direta do usuário.

Para a construção de processamento batch será utilizado o framework Spring Batch. Os Job's correspondentes aos processamentos serão embutidos em uma aplicação web isolados da aplicação principal.

O Spring Batch utiliza uma estrutura chamada de JobRepository para o armazenamento de metadados referentes aos Jobs. Os metadados podem ser armazenados apenas em memória RAM ou em banco de dados.

Durante as atividades de desenvolvimento deverá ser utilizado o armazenamento em memória RAM. Em ambientes de teste (desenvolvimento/homologação) e produção deverá ser utilizado o armazenamento em banco de dados.

5.2.8 Autorização/Autenticação

O Sistema de Cadastro e Permissão de Acesso (SCPA) foi desenvolvido pelo Ministério da Saúde no Departamento de Informática do SUS (DATASUS), com o intuito de unificar o cadastramento dos usuários aos sistemas WEB do Ministério da Saúde.

O SCPA disponibiliza um conjunto de webservices que serão consumidos para:

- Autenticação do usuário
- Recuperação de perfis de um usuário
- Recuperação das permissões de um perfil

O framework Spring Security será utilizado nas operações de autenticação e autorização.

O framework do DATASUS REST SPRING possui uma implementação padrão com os componentes de autenticação e autorização na camada WebService RESTful baseada no SCPA.

5.2.9 Paginação

Será utilizada a técnica de paginação sob demanda com o uso dos recursos de banco de dados e do framework de dados.

Basicamente, os repositórios deverão disponibilizar métodos que retornem uma instância de Page e recebam em seus parâmetros um objeto do tipo Pageable. O framework calculará automaticamente a quantidade total de objetos.

5.2.10 Envio de email

Utilizar os recursos disponibilizados pelo Framework. A geração do texto do e-mail deverá ser feita com o uso de templates definidos e construídos com o Velocity.

5.2.11 Relatórios

Os relatórios serão produzidos com o uso do framework Jasper reports. A geração será feita por classes de serviços. Para evitar problemas como o consumo elevado do heap space durante a etapa de Preenchimento (Fill Phase) em grandes relatórios, deve ser usado o recurso de virtualização do Jasperreports. Podem ser utilizados os tipos de virtualidade abaixo:

A definição do tipo de virtualizador a ser utilizado é responsabilidade do arquiteto do sistema. O consumo de recursos para relatórios com saída diferente de PDF deverá ser analisado com cuidado pelo arquiteto do sistema, pois algumas apis de exportação (ex: JexcelAPI) exigem que todo o conteúdo esteja em memória.

Será utilizada como linguagem de consulta a EJBQL. Para evitar problemas de consumo de memória, deverá ser utilizado o recurso de paginação. O número de objetos por páginas (jpa) será definido com preenchimento da propriedade **net.sf.jasperreports.ejbql.query.page.size**.

5.2.12 Teste unitário

Um teste unitário deve ser capaz de examinar o comportamento do código sob as mais variadas condições, ou seja, como o código deve se comportar se determinado parâmetro for passado (ou não), o que ele retorna se determinada condição for verdadeira, os efeitos colaterais que ele causa durante a sua execução, se determinada exceção é lançada, etc.

Os frameworks Junit e Mockito constituirão a estrutura básica para os testes unitários nos sistemas. Itens a serem cobertos por testes de unidade:

5.2.13 Logs

Será utilizado o framework Log4j para o registro de logs gerados pelo sistema.

Appenders indicados para uso:

FileAppender: Os responsáveis pela administração de servidores de aplicação redirecionam as informações enviadas para o console gravando em arquivos, com uma rotina de compactação e rodízio automático.

SMTPAppender: Utilizado para enviar as informações geradas pelo log para um endereço de e-mail que será acessado pela equipe responsável pela manutenção/evolução do sistema.

5.2.14 Trilhas de Auditoria

As trilhas de auditoria serão registradas automaticamente em tabelas previamente definidas com a equipe de administração de dados. Será necessário, porém, informar a sessão do banco de dados, as informações que serão registrados na trilha de auditoria.

O framework DATASUS já possui uma implementação que executa os procedimentos de recuperação dos dados de usuário e o IP do computador que executou a operação e atribuição a sessão do banco de dados. Para utilizar essa funcionalidade é obrigatório que o repositório extenda a classe que contém a implementação do registro de trilhas de auditoria.

5.2.15 Gerência de dependências

Os projetos deverão ser construídos utilizando gerência de dependências. Além de permitir um melhor controle dos artefatos e dependências de um projeto, o Maven permite que a construção do projeto seja feita de forma automatizada pela equipe de gestão de configuração.

Cada projeto e subprojetos deverá possuir um arquivo pom.xml na raiz do diretório raiz com as configurações do projeto e suas dependências.

O DATASUS disponibiliza os artefatos Maven através do repositório Nexus.

Este repositório fica localizado na URL:

http://repo1-maven.saude.gov.br/nexus/.

Para auxiliar e padronizar os projetos desenvolvidos no DATASUS, na criação dos templates iniciais dos projetos, serão utilizados arquétipos (archetypes) do Maven.

O archetype será atualizado de acordo com a arquitetura de referência e suas diretrizes.

5.2.16 API

É importante que quem consuma a API saiba do que se trata e quais os contratos que são firmados no envio e no recebimento de dados. O Swagger é uma das ferramentas mais importantes, que visa solucionar esse problema de documentação e outros.

O Swagger é uma ferramenta que pode ser usada em todo o ciclo de desenvolvimento de uma API REST, desde o design até os testes; e essa ferramenta está disponível para várias linguagens como PHP, C, Java entre outras.

6. VISÃO DE IMPLANTAÇÃO

As visões de implementação irão seguir a diretriz de referência para cada linguagem, observando sua tecnologia, versões e formas de implementação.

7. REFERÊNCIAS

- Patterns of Enterprise Application Architecture, Martin Fowler et al., http://www.martinfowler.com
- Repository, JavaBuilding, URL: http://www.javabuilding.com/academy/patterns/repository.html
- REST API Design RuleBook, Mark Masse, http://shop.oreilly.com/product/0636920021575.do OWASP Top 10 { 2013 - https://www.owasp.org/images/9/9c/OWASPT op102013P TBR:pdf