PROJETO GEOCAB

Documento de Arquitetura do Software

Versão 1.0

Histórico de Revisões

Data	Versão	Descrição	Responsável
10/10/2014	1.0	Criação do documento com informações essenciais e visão geral da arquitetura.	Vinicius Ramos Kawamoto

Sumário

1. Introdução

A finalidade deste documento é definir um modelo arquitetural para ser aplicado ao desenvolvimento do software para o Sistema GeoCAB, com uma visão macro dos requisitos arquiteturais e não funcionais para suportar a construção.

O objetivo deste documento é capturar e convergir decisões arquiteturais a serem aplicadas na construção de sistemas nesta plataforma. A arquitetura é formada por diversos padrões de projeto, principalmente, padrões Orientados a Objetos com destaque no mercado. Iremos destacar em cada parte da arquitetura o motivo da sua criação e o qual a sua influência para a criação de sistemas de alta coesão, mas com baixo acoplamento.

Este documento serve como um meio de comunicação entre o Arquiteto de Software e outros membros da equipe de projeto sobre as decisões significativas que forem tomadas durante o projeto.

1.1. Público Alvo

- Gerentes de Projeto;
- Equipe de projeto de arquitetura;
- Equipe de desenvolvimento de software;
- Engenheiros de software e testadores;
- Fornecedores a serem contratados para desenvolvimento e manutenção de Sistemas

1.2. Visão geral

Estão contemplados neste documento os padrões de projeto, componentes de software, plataforma de desenvolvimento, servidor de aplicação, banco de dados, frameworks e APIs. São também descritos as camadas de que é composto o modelo arquitetural, requisitos de segurança, requisitos de desempenho e requisitos de integrações.

A Figura 1 representa uma visão geral da arquitetura, com ilustração das camadas e componentes utilizados que são detalhados no decorrer deste documento.

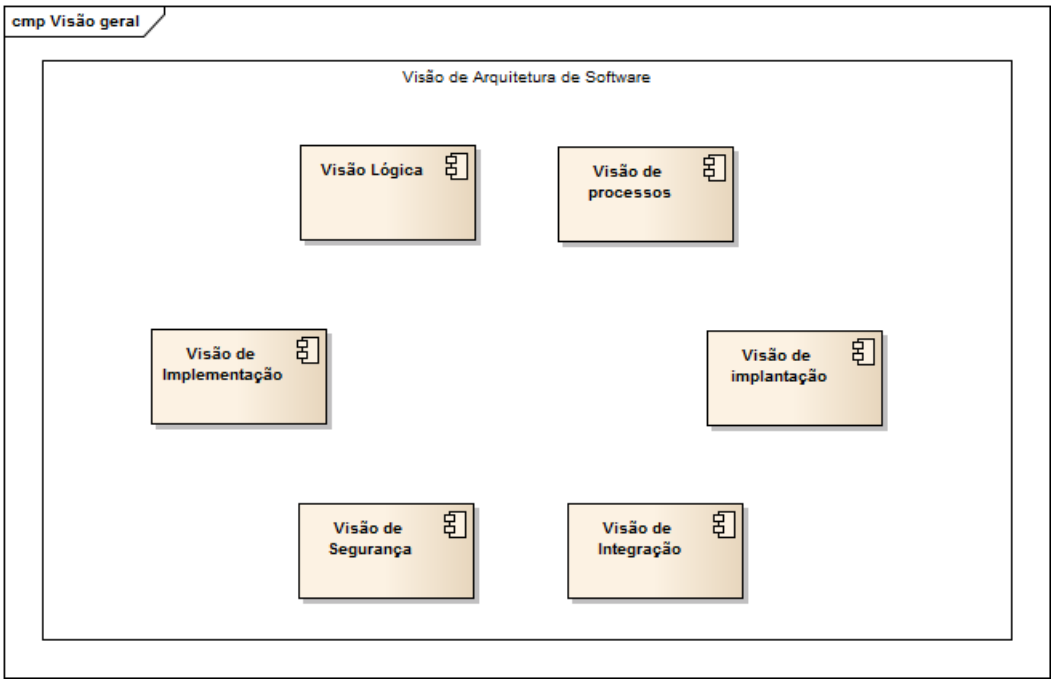

Figura 1 - Visão Geral da Arquitetura

2. Metas e Restrições de Arquitetura

Prover uma arquitetura de sistemas e de dados que atenda à necessidade proposta e que ao mesmo tempo seja flexível de forma a permitir a evolução do sistema.

Para a arquitetura proposta, foram considerados fatores como finalidade do sistema, tipo de usuários e ambiente de execução. Sendo assim, a arquitetura atende às seguintes características:

- **Modularidade**: Faz com que o sistema possua partes não acopladas, facilitando a possível substituição de componentes do mesmo. Essa característica se mostra muito importante, por se tratar de um aplicativo voltado para a Web, onde frequentemente surgem novas tecnologias ou até mesmo novas versões, que demandam reestruturação parcial do código.

- **Manutenibilidade:** Refere-se à facilidade, precisão, segurança e economia na execução de ações de manutenção nesse sistema. Através do uso do paradigma de programação orientado a objetos, será possível obter manutenibilidade, possibilitando alta coesão e baixo acoplamento entre os componentes do sistema,

o que resulta em um código mais organizado e de fácil entendimento. A arquitetura também provê artifícios para fácil modificação do software a fim de corrigir defeitos, adequar-se a novos requisitos, aumentar a suportabilidade ou se adequar a um novo ambiente.

- **Extensibilidade**: Analisando o objetivo do produto como um todo, é possível observar que a arquitetura deve possibilitar atualizações e extensões a novos módulos. Portanto, a arquitetura projetada tem o objetivo de facilitar a adição de novos módulos assim como evolução de funcionalidades existentes.

- **Reusabilidade**: Todo componente de infraestrutura da arquitetura, assim como classes que representam o domínio da aplicação são desenvolvidos observando um futuro reuso, favorecendo o tempo de produção e a qualidade do produto gerado.

- **Interoperabilidade:** Para atender aos requisitos de interoperabilidade é previsto para a comunicação com outros sistemas a utilização dos padrões *Web Map Service* (WMS) e *Web Feature Service* (WFS) para acesso e manipulação de dados geográficos.
- **Segurança:** A arquitetura permite atender as regras específicas de segurança tendo em vista os atributos de confidencialidade, integridade, disponibilidade e autenticidade dos dados e também dos acessos às funcionalidades do sistema, com uma visão sobre banco de dados, servidor e cliente.

- **Portabilidade**: Garantir que o sistema possa ser transferido de um ambiente para outro em diferentes condições de infra-estrutura (sistemas operacionais, versões de bancos de dados, etc). Como abordagem de portabilidade, a solução será implementada utilizando os princípios da plataforma *JEE (Java Enterprise Edition),* implementadas pelo *Spring Framework* que fornece uma API para programação de softwares corporativos e será executada em um servidor de aplicações que poderá ser independente de sistema operacional

- **Usabilidade:** Define a facilidade com que os usuários podem empregar o software a fim de utilizar uma funcionalidade específica e normalmente importante. Para tal, todo o design gráfico do sistema utilizará boas práticas providenciando a facilidade no aprendizado das funcionalidades, facilidade de memorização o que permite um

usuário não frequente saber utilizar o sistema assim como utilizar o sistema sem erros, e quando ocorrer, o sistema e o usuário saber se recuperar.

3. Visão de Integração

O GeoServer, servidor que integra diversos repositórios de dados geográficos, contém informações de camadas, que serão utilizadas no sistema GeoCAB evitando a duplicidade de informação, assim como a garantia de consistência dos dados, uma vez que a base de camadas é unificada.

Dados de camadas do GeoServer como nome, legenda e coordenadas, serão publicados no GeoCAB no formato de serviço.

3.1. Abordagem

O projeto prevê a utilização de padrões de referência do GeoServer para acesso e manipulação de dados geográficos. Estes padrões de acesso e manipulação atuarão fisicamente na Itaipu Binacional e o GeoCAB atuará como cliente, tendo seu ambiente de produção também na Itaipu Binacional.

3.2. Padrões de Integração

A solução prevê utilizar a arquitetura de interoperabilidade entre sistemas disponível através do GeoServer pelos padrões *Web Map Service* (WMS) e *Web Feature Service* (WFS).

O seguinte conjunto de tecnologias serão utilizadas para a integração dos dados do GeoServer com o GeoCAB:

Tipo do Componente	**Tecnologia**
Interconexão :: *Camada de Aplicação*	HTTP/1.1
Interconexão :: *Camada de Rede/Transporte*	TCP/IP
Segurança :: *Comunicação de DAdos*	TLS/SSL
Segurança :: *Criptografia*	SHA-512
Organização e Intercâmbio :: *Tratamento e Transferência*	XML

Padrão para acesso de informação geográfica	WMS
Padrão para acesso e manipulação de informação geográfica	WFS

4. Visão de Lógica

A descrição da visão lógica da arquitetura prevê as classes mais importantes, sua organização em pacotes e subsistemas de serviço, e a organização desses subsistemas em camadas. Descreve também as realizações de caso de uso mais importantes como, por exemplo, os aspectos dinâmicos da arquitetura.

4.1. Visão Geral

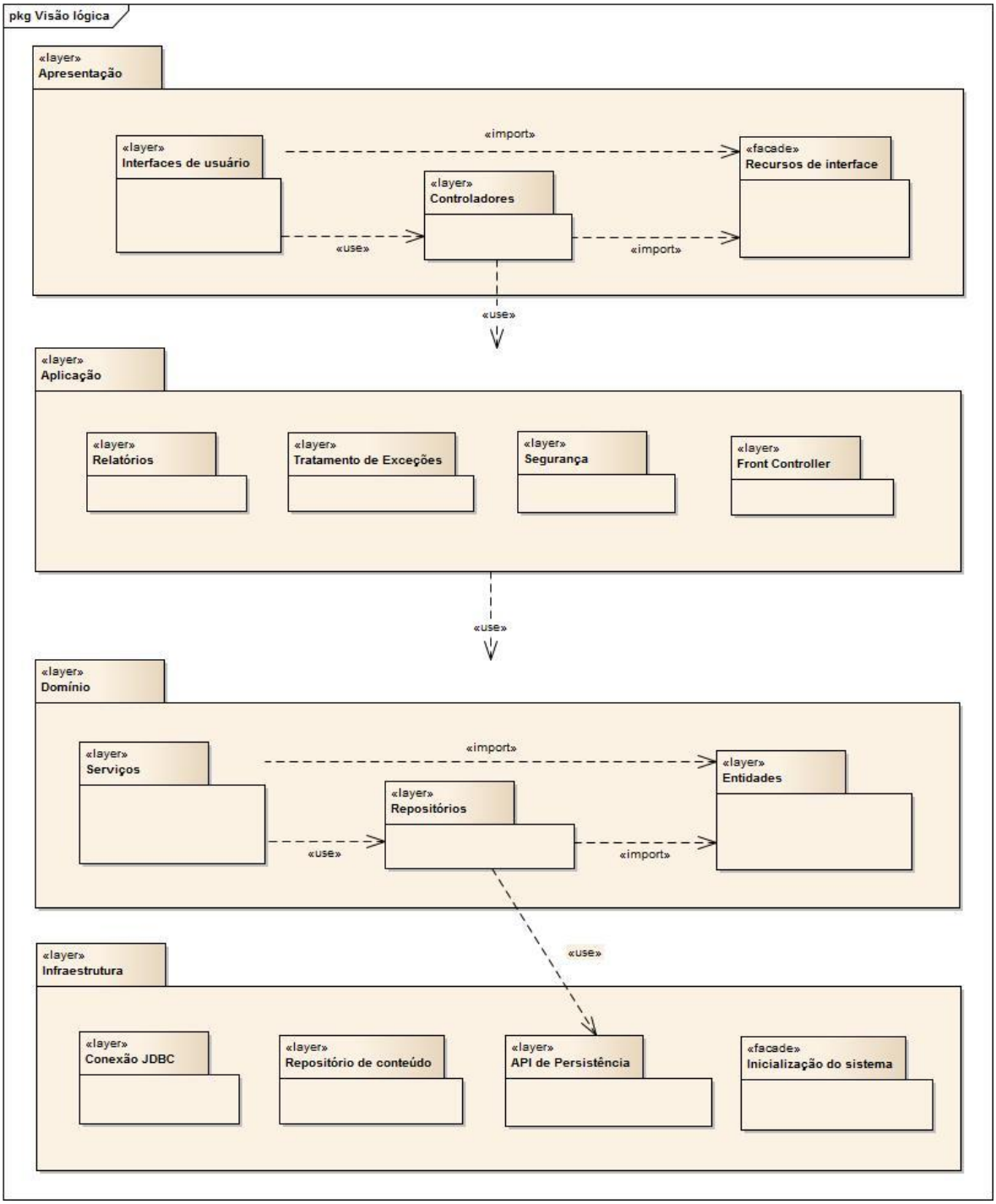

Figura 2 – Visão geral das camadas

A arquitetura foi projetada prevendo 4 camadas, que seguem o padrão de arquitetura *Domain-Driven Design*, definido por Eric Evans e com padrões de projetos definidos por Martin Fowler:

- **Camada de Apresentação:** responsável por agrupar os arquivos de interfaces de usuário, além de possuir a camada de controladores dos eventos de telas.

- **Camada de Aplicação:** coordena as implementações de suporte. Sua responsabilidade é de trabalhar com os objetos de domínio e não manter o estado de domínio, ou seja, não contém regras de negócio. Pode manter um estado de determinada transação, porém ela possui uma ligação muito forte com a camada de domínio (*Domain*), além de realizar o tratamento de exceções e configurações de segurança e expor os serviços do domínio através de um barramento central (*front controller*).

- **Camada de Domínio:** concentra toda a regra de negócio da aplicação. Segundo Eric Evans, a camada de domínio é o coração de um software de negócios. Dentro desta camada se encontram os repositórios que manipulam os dados, as entidades que representam os estados e comportamentos do domínio e também as classes de serviços que exportam o acesso à camada de domínio.

- **Camada de Infraestrutura:** considerada de mais baixo nível. É responsável pelas interações com a infraestrutura técnica. Nesta camada são colocadas as tarefas de interface com o sistemas de e-mail, configurações iniciais, banco de dados, sistemas de arquivos e outros objetos de infraestrutura, ou seja, dar suporte tecnológico para as demais camadas.

A Figura 3 representa algumas classes das entidades da camada de domínio. Nos exemplos, são utilizadas classes normais, que tem relacionamentos de agregação, composição e em alguns casos de herança. Para representação de dados enumerados, é utilizado o recurso *Enums* do Java 5+. *Enums* é utilizado em entidades imutáveis (ex.: ResolucaoMapa) ou em entidades onde uma mudança pode inferir diretamente na regra de negócio (ex.: 1:70M, 1:35M).

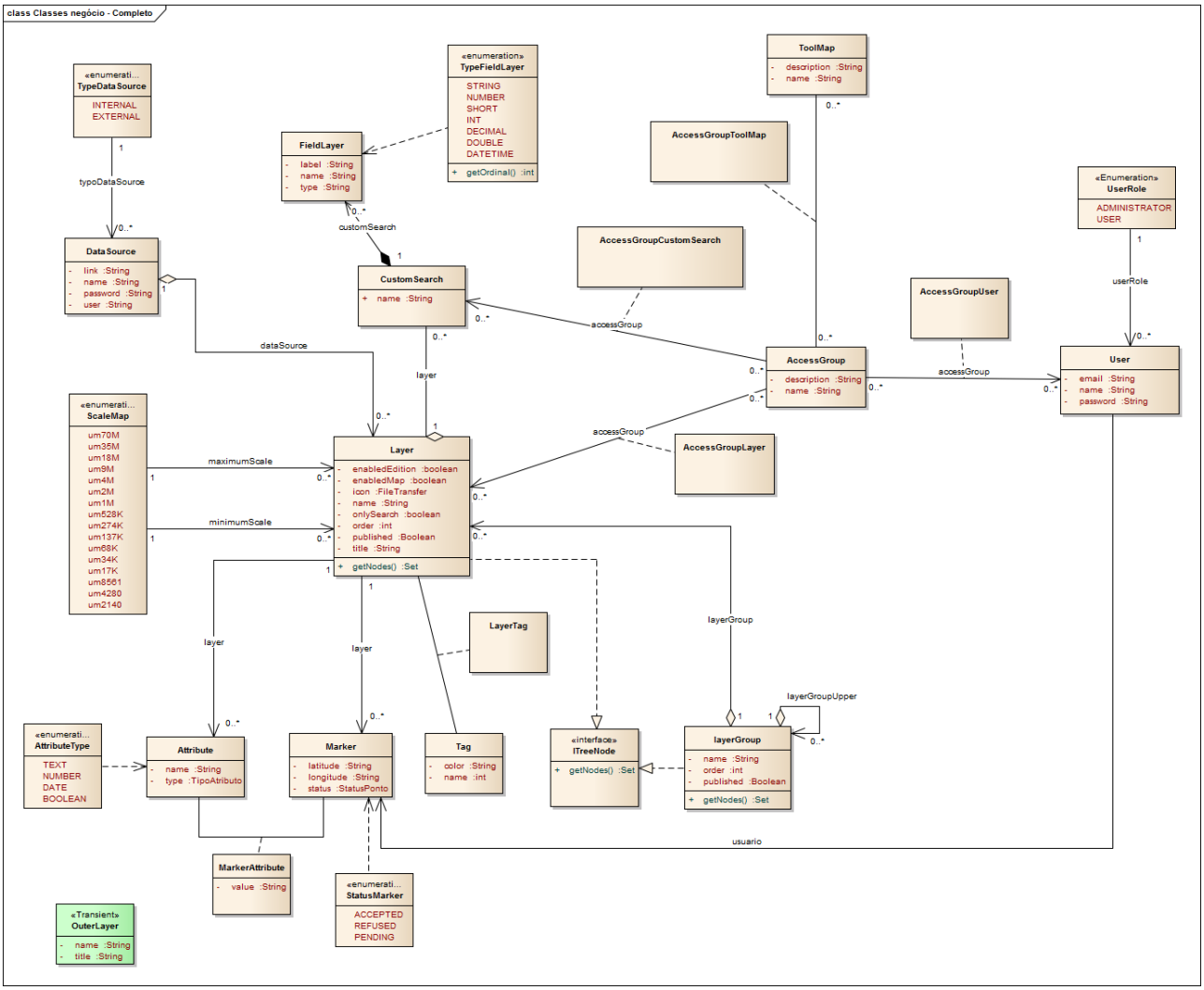

Figura 3 – Diagrama de classes do GeoCAB

4.2. Visão Serviços

A arquitetura exporta os serviços, seguindo as boas práticas de SOA - *Service-Oriented Architecture,* cujo princípio fundamental prega que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas no formato de serviços.

A Figura 4 é um exemplo de serviço modelado para atender aos requisitos do sistema GeoCAB.

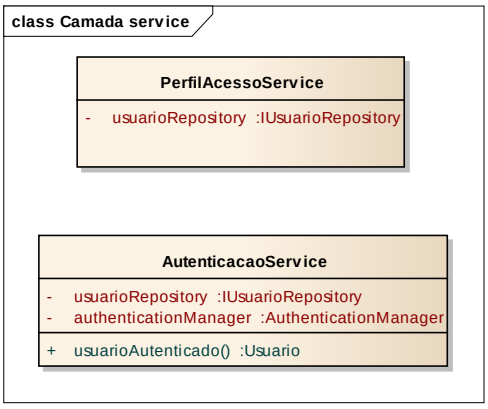

Figura 4 – Diagrama de classes de serviços do sistema GeoCAB.

4.3. Visão de banco de dados e acesso a dados

Na arquitetura proposta, é utilizado um banco de dados relacional para persistência dos dados do domínio da aplicação. Para melhor organização dos dados e facilidade de manutenção evolutiva, a arquitetura utiliza um banco de dados que tem a seguinte hierarquia: banco -> esquema(s) -> tabela(s). Para cada módulo do sistema, será utilizado um esquema no banco. A seguinte estrutura de esquemas foi prevista incialmente:

- **Esquema "geocab_audit":** Contém as tabelas de todos os módulos, com finalidade de registrar o log de todas as manipulações realizadas por usuários no sistema. Como o próprio nome sugere, o objetivo deste esquema é realizar a auditoria nos dados da aplicação.
- **Esquema "geocab":** Contém as tabelas referentes ao sistema GeoCAB.

Quanto ao acesso dos dados por parte do componente servidor na arquitetura proposta, os dados são manipulados na forma de repositórios. Cada entidade persistente tem seu repositório e fornece as classes de serviços, métodos de manipulação e consulta de dados. A Figura 5 é um exemplo de repositório modelado pertinente ao sistema GeoCAB.

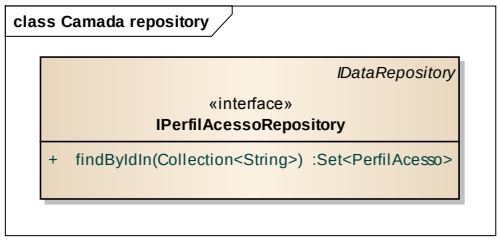

Figura 5 – Diagrama de classes dos repositórios do sistema GeoCAB.

5. Visão de Processos do Sistema

Esta seção descreve a decomposição do sistema em processos leves (*threads* de controle únicas) e pesados (agrupamentos de processos leves).

Conforme a Figura 6, na arquitetura proposta todas as requisições entre cliente e servidor são controlados por *threads*. Estas *threads* são controladas pelo próprio *container web* do servidor de aplicações. Neste caso, nenhum controle adicional é necessário.

Um outro componente importante na arquitetura que consome recurso é o contexto de injeção de dependência gerenciado pelo *Spring Framework*, no qual utiliza de *threads* para gerenciar a camada de acesso a dados (*JPA + Hibernate*), aspectos e os filtros de segurança.

Quanto a comunicação entre o servidor e o banco de dados, os processos são gerenciados por um *pool* de conexão, que gerencia as conexões conforme a concorrência no sistema.

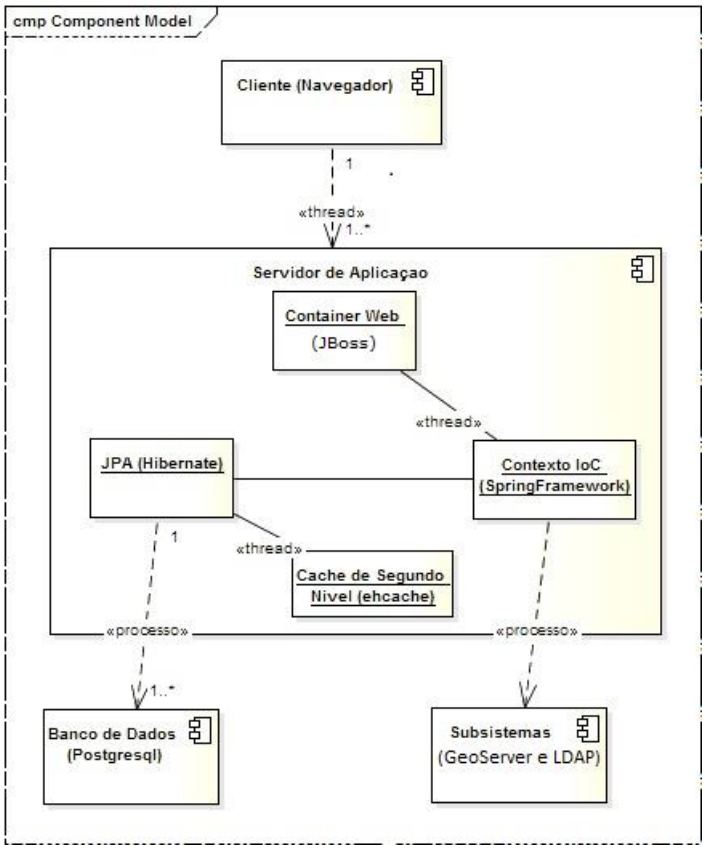

Figura 06 – Visão de processos do sistema

6. Visão de Implementação Web

Essa seção descreve a estrutura geral do modelo de implementação e a decomposição do software em camadas e subsistemas.

Foram considerados os requisitos não funcionais, normas e especificações, regras de negócio, além de boas práticas de desenvolvimento de software para elaboração da arquitetura.

6.1. Visão Geral

A arquitetura foi projetada prevendo 4 camadas conforme o modelo arquitetural definido pelo *Domain-Driven Design,* de Eric Evans. Para cada camada são utilizados padrões de projetos desenhados por Martin Fowler, apresentados no catálogo *Patterns of Enterprise Application Architecture*.

Domain-Driven Design trata-se de uma abordagem de *design* de *software* que ajuda a construir aplicações que refletem a compreensão e a satisfação das exigências do negócio,

abordando uma série de conceitos e técnicas com foco no domínio do software.

6.2. Abordagem

Como solução base para a implementação da arquitetura proposta, será utilizado o *Spring Framework* versão 4.0.4.RELEASE, um componente de servidor de código aberto composto por um container de inversão de controle para a plataforma Java. Este framework segue as especificações JEE 6 homologadas da plataforma Java, mantendo a simplicidade e totalmente modularizado.

A técnica de inversão de controle permite que a sequência das chamadas aos métodos seja invertida em comparação com a programação tradicional, ou seja, ela não é definida diretamente pelo desenvolvedor. Este controle é delegado a uma infraestrutura de software que realiza a chamada ao *container* do *Spring Framework* para que possa tomar controle sobre a execução. Basicamente, o desenvolvedor "solicita" uma instância válida dos componentes necessários para o desenvolvimento, e o *container* é quem é o responsável por devolver uma instância válida (já configurada pronto para uso).

Outra abordagem para implementação da arquitetura é a utilização da programação orientada a aspectos para resolver problemas de transação (implementado pelo *transactionManager* via anotação *@Transactional*), segurança (implementado pelo módulo *SpringSecurity*, via anotação *@PreAuthorize*) e tratamento de exceções globais (implementado pelo aspecto customizado *br.gov.itaipu.geocab.application.aspect.ExceptionHandlerAspect*). Como solução, o próprio *Spring Framework* contém um módulo que permite a fácil configuração dos aspectos.

Para implementação do cliente é utilizado um *framework* MVW (*Model-View-Whatever*) para *JavaScript* que permite o desenvolvimento ágil e voltado para performance. O *framework* adotado é o AngularJS mantido pelo Google no qual permite construir as telas do cliente independente de processamento servidor o que possibilita o servidor ser desenvolvido orientado a serviços e o cliente como consumidor destes, através de um barramento de serialização de dados entre Java e *JavaScript*.

As tecnologias envolvidas são representadas na figura 7.

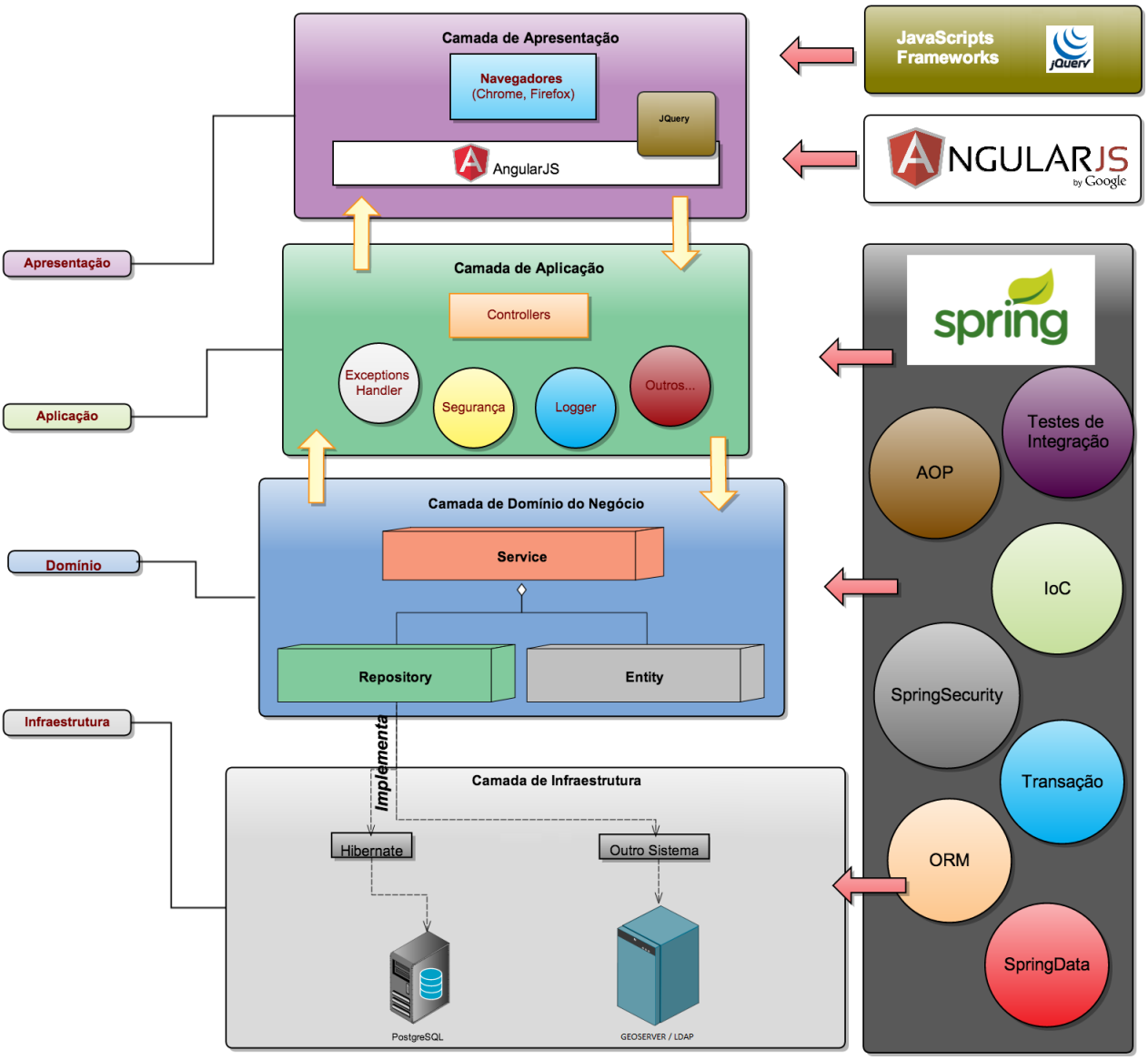

Figura 07 – Visão da arquitetura com uma perspectiva das camadas e com as tecnologias.

6.3. Implementação das Camadas

6.3.1. Camada de infraestrutura e acesso a dados

Para atender a camada de acesso a dados encapsuladas por repositório e alimentado pela a camada de infraestrutura (Figura 2 e 6), a persistência dos objetos de domínio é realizada através da técnica de mapeamento de objeto relacional (ORM) para resolver problemas entre a programação orientada a objetos e o banco de dados relacional. Como solução, foi optado por utilizar os seguintes componentes:

- ***Spring Data***: Módulo do *Spring Framework* para acesso a dados. Este módulo traz facilidade de desenvolvimento da camada de acesso a dados com total desacoplamento e voltada para performance, pois o desenvolvedor apenas implementa as consultas otimizadas para pesquisa em uma *interface* Java através de *annotations (@Query)*. Operações de inserção, alteração e exclusão já vem pronto, além de ter a técnica de fazer consultas através dos nomes do método (ex.: *findById* – executa um *select* na tabela, procurando por Id). Com este módulo, recursos de paginações e ordenações são facilmente implementados a partir dos parâmetros *Pageable* para realização de paginação e *Sort* para a realização de ordenação.

- ***JPA - Java Persistence API*:** Como solução ORM foi adotado a especificação JPA na versão 2.0 integrada ao *Spring Data*. Como abordagem da arquitetura, o *EntityManager* é gerenciado pelo *Spring Data* e este abstrai para a camada de serviços como métodos no formato de repositórios (Figura 5).

- ***Hibernate*:** Como implementação da JPA, foi adotado o *Hibernate* na versão 4. Um excelente *framework* que permite o carregamento Lazy ou EAGER dos dados, cascateamento entre tabelas e mapeamento de todos os tipos de relacionamentos (ex.: Um para muitos, muitos para muitos etc.)

6.3.2. Camada de domínio e serviços

Na camada de serviços são realizadas as transações do sistema, através dos aspectos do *Spring Framework*, facilmente programada através da *annotation* @Transactional. Esta *annotation* utiliza a estratégia propagação requerida (*Propagation.REQUIRED*), que consiste em utilizar uma transação existente caso já exista uma, caso não, criar um nova para realizar a operação. Nos serviços de consulta, adicionamos as transações como somente leitura (*readOnly=true*), evitando modificações acidentais e economizando recurso do servidor, uma vez que a transação é somente leitura.

É na camada de serviços que a segurança é aplicada através do módulo *Spring Security* do *Spring Framework*. Todo serviço seguro é vinculado a uma permissão e é implementada através da *annotation @PreAuthorize.* Mais detalhes ver Visão de Segurança.

Para a camada cliente, os serviços são exportados através de um barramento que publica classes Java no formato de classes *JavaScript* chamado DWR - *Direct Web Remoting*. Na arquitetura, foi utilizado o recurso da *annotation @RemoteProxy* que faz uma introspecção na classe Java, verificando os métodos públicos e gera URLs dinamicamente expondo a classe Java no formato de classe JavaScript para enfim o cliente que executa um navegador, consumir os serviços de forma transparente.

6.3.3. Camada de Apresentação

A camada de apresentação ou *frontend* é executada no componente cliente e é implementada utilizando *JavaScript*, HTML5 e CSS3. Para integração com o componente de servidor e também facilitar o desenvolvimento das interfaces garantindo performance é utilizado o framework AngularJS, que traz recursos como: *binding* entre o dado e a tela, infraestrutura para criação de componentes customizados (*directives*), separação entre dado e tela, *cache* de telas, separação entre tela e camada de controle e injeção de dependência.

As telas são construídas em HTML5 com um tema elaborado em CSS3 que faz *binding* com um escopo mantido por uma classe *JavaScript* controladora, que quando processado pelo navegador, no HTML é substituído os valores dinâmicos por reais além de ficar monitoramento mudanças no escopo para refletir na tela. A classe controladora faz as devidas chamadas (“Ajax”) assíncronas ou quando específico, de forma síncrona para o servidor através do DWR e alimenta o estado no escopo para refletir na tela (Figura 8).

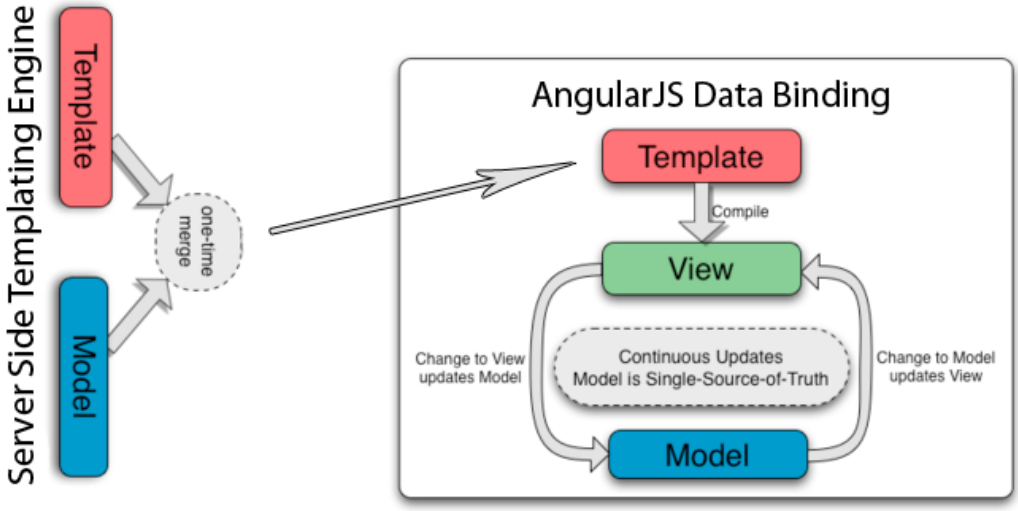

Figura 08 – Fluxo de execução da camada de apresentação com AngularJS.

6.4. Código Fonte

O código fonte do sistema está organizado em um projeto que contém as funcionalidades e os testes de integração (Figura 9).

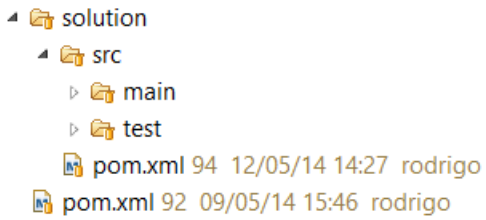

Figura 09 – Projeto *solution*.

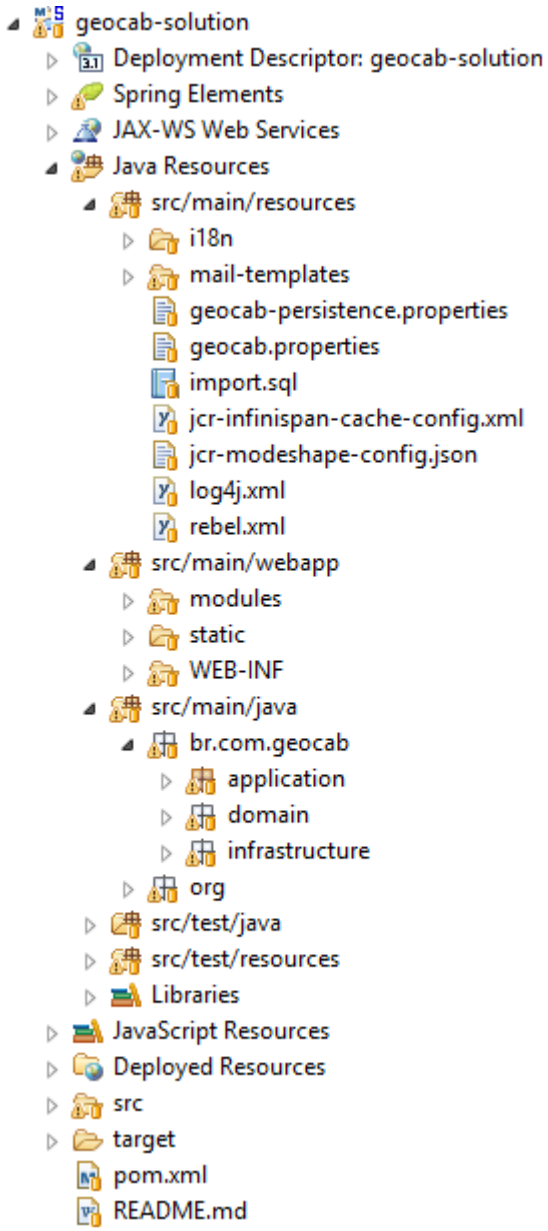

Figura 10 – Projeto *solution* expandido

O projeto será construído seguindo a convenção da ferramenta livre para compilação e empacotamento, *Apache Maven*. Logo, toda raiz de projeto contém um arquivo *pom.xml* com as configurações e dependências do projeto, assim como por convenção, o código fonte Java é aplicado em *src/main/java* e arquivos não Java como arquivos de configuração e de texto são armazenados em *src/main/resources*.

A pasta *test* contém a implementação dos testes de integração. O tipo do projeto foi configurado como um projeto Java normal no Maven. O projeto *solution* é chamado assim pois é responsável por gerar a solução final do projeto, no caso, um arquivo .WAR com todos os arquivos necessários para implantação. Este projeto está devidamente configurado no *Maven* como um projeto *Java Web*. Por padrão, este projeto por ser importado no eclipse como um projeto *Dynamic Web*.

A figura 10 contempla o projeto *solution* aberto e como a arquitetura é refletida no código, como por exemplo as camadas de *aplicação*, *domínio* e *infraestrutura*. O pacote "geocab" representa o agrupamento de classes necessárias para as funcionalidades de configuração de camadas, grupo de camadas, fonte de dados geográficos, grupos de acesso, pesquisas personalizadas.

7. Visão de Implementação Mobile

Essa seção descreve a estrutura geral do modelo de implementação mobile para plataforma *Android* e *IOS*.

Foram considerados os requisitos não funcionais, normas e especificações, regras de negócio, além de boas práticas de desenvolvimento de software para elaboração da aplicação.

7.1. Android

Para implementação da aplicação é utilizado um padrão MVC sem utilização de framework, para manter o código mais puro, buscando a performance e com intuito de deixar o código com uma manutenibilidade melhor para possíveis alterações.

Os serviços utilizados no *Android* é realizado por meio de serviços REST e foi utilizado uma biblioteca chamada *Volley*, uma biblioteca atual e desenvolvida pela *Google* com código aberto. Com ela é possível obter inúmeros benefícios como: agendar automaticamente as solicitações da rede, controlar o cache das requisições dos dados, principalmente das imagens, evitar o trabalho com AsyncTask e utilização de Threads secundárias, deixando o trabalho mais rápido, fácil e consistente.

A versão mínima para aplicação é a versão 4.0 e para utilizar a aplicação exige algumas permissões definidas no arquivo de configurações do *Android*.

- android.permission.INTERNET
- android.permission.ACCESS_NETWORK_STATE

- android.permission.GET_ACCOUNTS
- android.permission.USE_CREDENTIALS
- android.permission.ACCESS_FINE_LOCATION
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.VIBRATE
- android.permission.READ_PHONE_STATE

7.2. IOS

8. Visão de Segurança

A segurança na arquitetura está relacionada com a proteção de um conjunto de informações, no sentido de preservar o valor que possuem para uma pessoa ou uma organização. As características básicas da segurança da informação são os atributos de confidencialidade, integridade, disponibilidade e autenticidade.

8.1. Autenticação e Autorização

A segurança nesta aplicação é garantida por meio da utilização de controle de usuários usando gerenciamento de permissões e controle de sessão e foi utilizado o modelo RBAC (*Role-based access control*) que descreve a criação de permissões baseadas em papéis. Esta é uma abordagem para restringir a autenticação no sistema somente para usuários autorizados e realizar a autorização utilizando as permissões de todos os papéis que o usuário pode ter.

A segurança na arquitetura pode ser observada de 2 formas: Uma visão global e uma visão por funcionalidade. Regras de seguranças globais são configuradas no arquivo `geocab-security-context.xml` que define quais URLs estão bloqueadas ou abertas além de configurar o mecanismo de autenticação. Se um usuário anônimo tentar acessar o sistema, ele terá acesso apenas à tela de autenticação.

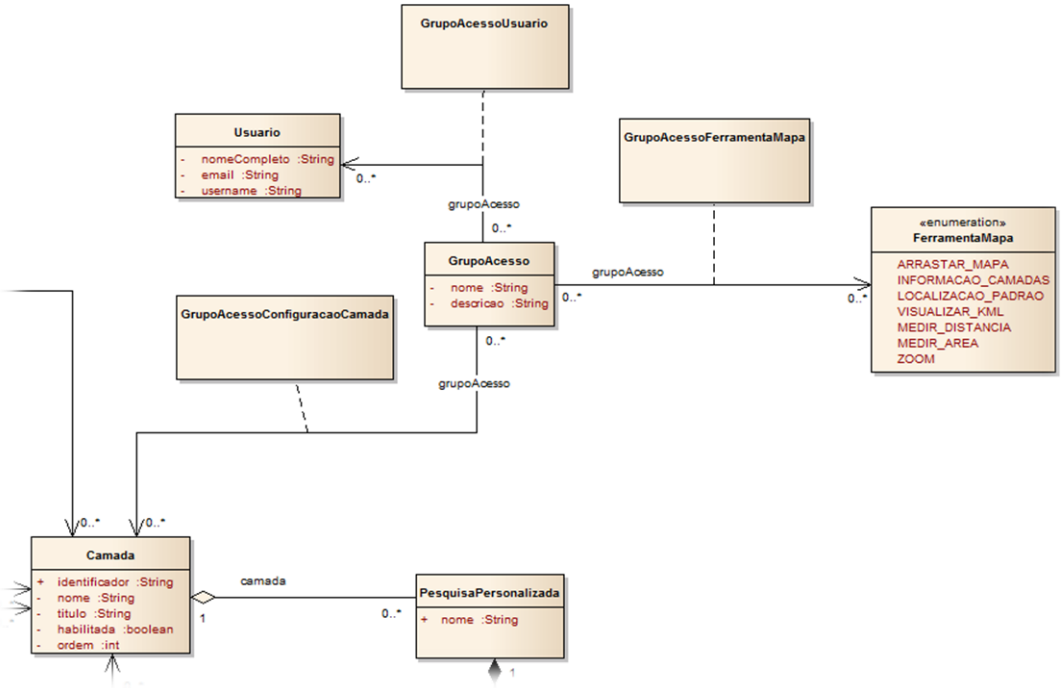

Figura 11 – Modelo de permissões e perfis de acesso.

A Figura 11 representa o modelo a ser persistido no banco de dados para grupos de acesso e seus relacionamentos. Basicamente, um usuário pode ter vários grupos de acesso e cada grupo de acesso pode ter vários usuários. A visualização de determinadas camadas e consulta de dados nelas ou a permissão para utilizar ferramentas específicas do mapa interativo só será possível se o grupo de acesso, que contém usuários, está associado à camada ou ferramenta.

Diferente da permissão de visualização e manipulação de dados por determinados grupos de acesso, existem ainda as permissões por funcionalidades da solução. Para acesso a funcionalidade apenas dois perfis existirão a nível de código com base no grupo LDAP do usuário: administrador e público. O administrador tem acesso a todas as funcionalidades do sistema e o público apenas ao mapa interativo.

8.2. Segurança contra ataques

Para a garantia de segurança do sistema transacionar informações com os usuários de forma segura, o acesso deverá será realizado através da tecnologia chamada "SSL" (*Secure Sockets Layer*) que criptografa os dados transmitidos entre um navegador e um

servidor da web.

Em uma conexão SSL, há mais privacidade e segurança do que em uma conexão da web sem criptografia. Com isso, há redução do risco de que as informações sejam interceptadas e usadas de forma indevida por terceiros, através de técnicas de ataque como por exemplo a *man-in-the-middle* (homem no meio, em referência ao atacante que intercepta os dados), uma forma de ataque em que os dados trocados entre duas partes são de alguma forma interceptados, registados e possivelmente alterados pelo atacante sem que as vitimas se apercebam.

Esta arquitetura esta prevista para ser executada utilizando o protocolo HTTP, com SSL, utilizando o esquema seguro HTTPS. O *container Web* deverá ser devidamente configurado para suportar tal segurança. Essa configuração é prevista no artefato “Plano de Implantação”.

9. Visão de Implantação

Para a arquitetura projetada, alinhado com os requisitos não funcionais pertinentes a usuários concorrentes e velocidade de resposta, as seguintes configurações são mínimas sob uma visão de rede, servidor, cliente e banco de dados.

9.1. Componentes de rede

Componente	Versão/Capacidade
Camada de Aplicação	HTTP/1.1
Camada de Rede/Transporte	TCP/IP
Velocidade da rede entre cliente e servidor mínima	10Mb/s
Velocidade da rede entre servidor e banco de dados mínima	100Mb/s

9.2. Componentes do servidor

Componente	Versão/Capacidade
Sistema operacional	Distribuição Linux (Recomendado distribuições Debian 7+)
Servidor de aplicação (Container Web)	Tomcat 8.0.12
Memória dedicada ao servidor de aplicação	4 Gb
URI Encoding do servidor de aplicação	UTF-8
Máquina virtual Java	JRE 1.8.0_20
Frequência do processador	2.8 Ghz

Memória RAM	8 Gb

9.3. Componentes do banco de dados

Componente	Versão/Capacidade
Sistema operacional	Distribuição Linux (Recomendado distribuições Debian 7+)
Banco de dados	PostgreSQL v9.2.7
Memória RAM	4 Gb
Frequência do processador	2.8 GHz

9.4. Componentes do cliente

Componente	Versão/Capacidade
Sistema operacional	MS Windows XP/Vista/7/8 Distribuição Linux Ubuntu 11 Apple Mac OS 10.7
Navegador	Google Chrome v34 Mozilla Firefox 24 ESR Internet Explorer 11
Memória RAM	2 Gb
Frequência do processador	1.6 GHz

A Figura 12 representa o digrama de implantação, contemplando todos os componentes e a comunicação entre eles.

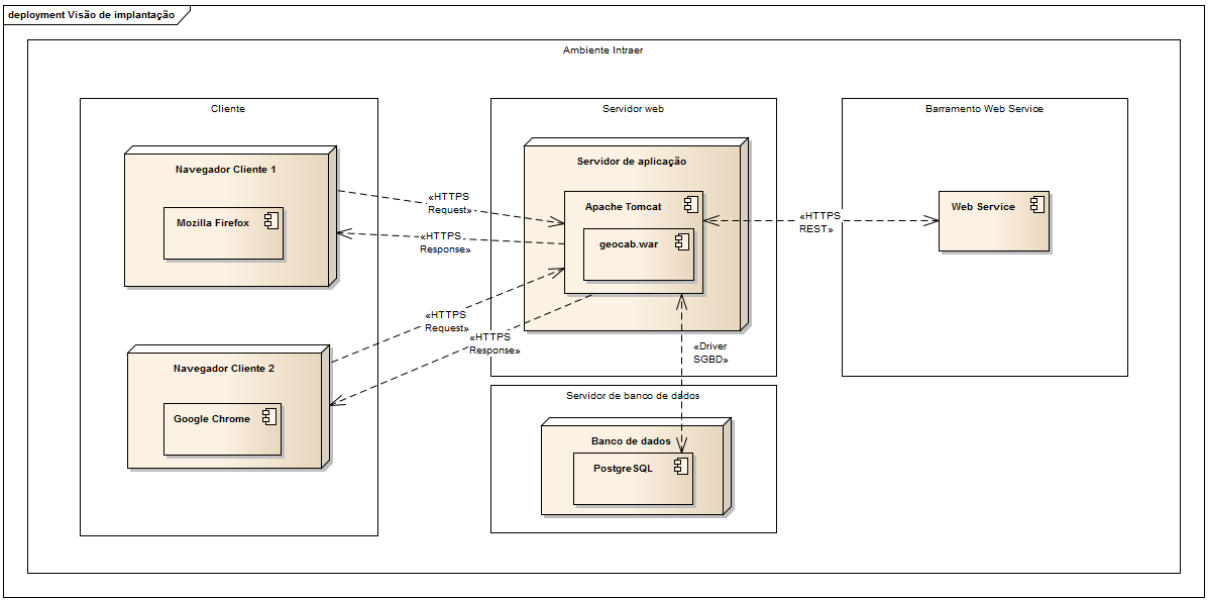

Figura 12 – Diagrama de implantação

10. Referências

EVANS, Eric. Domain-Driven Design: Tackling Complexity in the Heart of Software.

FOWLER, Martin. Patterns of Enterprise Application Architecture

FREEMAN, Eric T; ROBSON, Elisabeth; BATES, Bert; SIERRA, Kathy. Head First Design Patterns.

GAMMA, Erich. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos.

JOHNSON, Rod. Professional Java Development with the Spring Framework.

LEA, Doug; ALEXANDER Christopher. An Introduction for Object-Oriented Designers.