CLDF - ASSEL

Documento de Arquitetura de Software

Wedson Quintanilha da Silva

Versão 1.1.0, 27/07/2021

Índice

Histórico de Revisões

Data	Versão	Descrição	Autor	Revisor
12/07/2021	1.0.0	Criação do documento	Wedson Quintanilha da Silva	Vitor Santos Ferreira
27/07/2021	1.1.0	Remover o módulo Segurança	Wedson Quintanilha da Silva	Vitor Santos Ferreira

1. Introdução

Este documento visa descrever a arquitetura lógica e física do projeto ASSEL.

1.1. Definições, Acrônimos e Abreviações

- **SSO**: Single Sign-On, conceito de autenticação única e centralizada para várias aplicações
- **ARTEFATOS**: Documentos em geral, em diversos formatos que devem ser catalogados e ter o acesso

1.2. Documentos de Referência

Lista de documentos que foram utilizados como referência na elaboração da arquitetura de software:

- Proposta técnica

2. Objetivo do Documento

O objetivo deste documento é apresentar ao cliente a descrição da arquitetura do projeto.

3. Visões da Arquitetura

3.1. Critérios de Avaliação Arquitetural

Não aplica.

3.2. Arquitetura imposta pelo cliente

Para este módulo, o CLDF determinou que os requisitos de segurança sejam tratados pelo mesmo módulo "Catálogo de Artefatos" e não em um microsserviço separado e exclusivo para o domínio de Segurança.

3.3. Arquiteturas descartadas

Não aplica.

3.4. Visão Geral das Camadas Arquiteturais

Em arquitetura de sistemas, damos o nome de camada a uma estrutura lógica de componentes de um software que interagem entre si para cumprir um objetivo específico, responsabilizando-se pela execução de uma dada tarefa ou um conjunto de tarefas semelhantes. Em alguns casos, esse agrupamento pode não ser apenas lógico, mas sim assumir um caráter físico. Isso ocorre quando uma camada possui a habilidade de ser executada separadamente do restante da aplicação.

A arquitetura baseada em múltiplas camadas oferece um modelo flexível e reutilizável para o desenvolvimento de software. Ao se dividir uma aplicação em camadas, evitamos o forte acoplamento entre os componentes, de forma que a maioria das alterações necessárias possam ser feitas de maneira mais isolada e pontual, reduzindo o esforço necessário à manutenção do sistema.

Na figura abaixo, apresentamos uma visão simplificada das camadas que compõem o sistema e a forma como elas se relacionam. Nas seções a seguir, detalhamos conceitualmente cada camada e enumeramos as ferramentas e plataformas que serão utilizadas para sua construção.

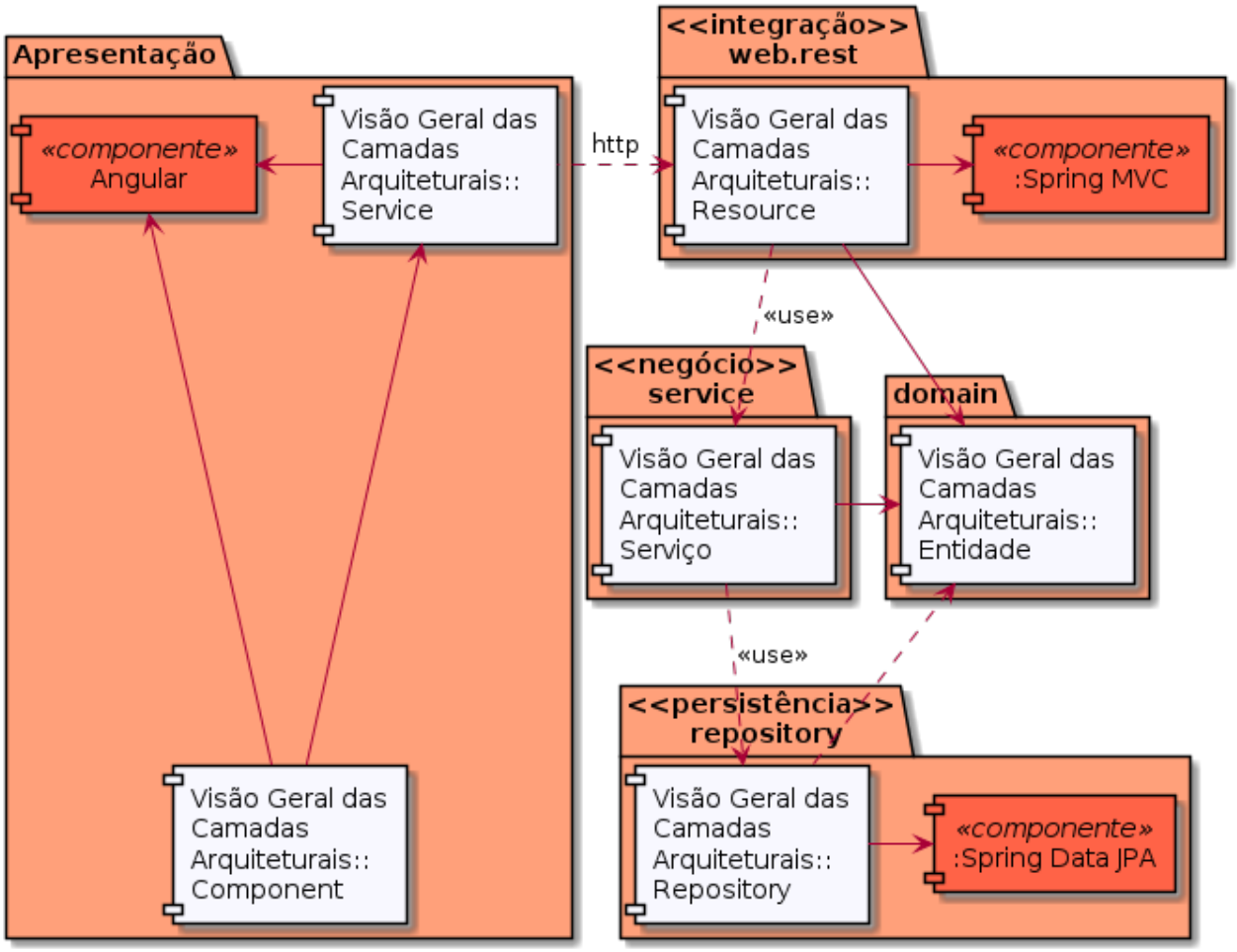

Imagem 1. Visão Geral das Camadas Arquiteturais

3.5. Camada de Apresentação

A camada de apresentação é responsável por fazer a lógica de construção das páginas para serem exibidas pelos usuários, tratar os eventos do browser, como cliques, e gerenciar o fluxo de execução do sistema.

3.6. Camada de Negócio

A camada de negócios é responsável pela implementação lógica da aplicação. Ela expõe os serviços para a camada de apresentação por meio de uma interface bem definida e obtém as informações necessárias para mostrar ao usuário por meio da Camada de Persistência.

3.7. Camada de Persistência

A camada de persistência é responsável pela lógica de acesso ao banco de dados e pelo mapeamento dos dados em entidades representativas. O objetivo em mapear o banco de

dados em entidades representativas ao sistema é diminuir a diferença semântica entre o modelo abstrato do banco e o problema do mundo real.

Para prover um melhor desempenho nas consultas, será utilizado o Elasticsearch para indexar as informações dos catálogos relacionados aos documentos gerenciados pela aplicação, possibilitando pesquisas textuais dentro dos atributos que compõem o domínio do catálogo sem prejudicar a performance do sistema.

Como esta é uma arquitetura baseada em microsserviços, para garantir a continuidade da operação caso algum microsserviço falhe ou fique fora do ar por algum motivo qualquer, a **recomendação** é que cada microsserviço tenha o seu banco de dados exclusivo. No entanto, caso isso não seja possível, a alternativa é manter um schema por microsserviço em separado e, caso isso também não seja possível, o projeto do banco de dados deve prever que as tabelas envolvidas em um domínio de negócio atendido por um microsserviço específico não sejam relacionadas com tabelas de outro microsserviço, apenas entre tabelas do mesmo domínio negocial atendido por um único microsserviço.

3.8. Camada de Serviços

A camada de serviços encapsula diversos serviços que são providos às outras camadas da aplicação.

Os serviços são responsáveis por realizar as regras de negócio e **estes são organizados por domínio negocial** e serão acessados por meio de API's REST.

O acesso aos serviços serão efetuados mediante a presença do token JWT (jwt-token) de autenticação válido, presente no cookie ou no header da requisição.

As API's presentes nesta camada serão organizados conforme padrões e princípios estabelecidos na especificação RFC7231, onde **resumidamente** temos:

- Uso apropriado dos verbos GET, PUT, POST, DELETE e PATCH conforme ações desejadas:

- **GET**: Usado para recuperar um recurso na Api;
- **PUT**: Usado para substituir, de forma idempotente, um recurso inteiro enviado para Api;
- **POST**: Utilizado para inserir um novo recurso enviado para a Api;
- **PATCH**: Utilizado para modificar partes de um recurso existente na Api;
- **DELETE**: Utilizado para apagar um recurso na Api.

- Uso correto dos Status de respostas (código de status) conforme as operações realizadas, observando e diferenciando erros de negócio de erros de sistema;
- Definição dos URI's (endpoints) de forma adequada, observando os relacionamentos entre os conceitos de negócio adequadamente;
- Versionamento correto adotando os prefixos para o caso de haver a necessidade de alterações nos comportamentos ou na estrutura de dados enviada ou recebida a partir de um serviço.

Exemplo:

Versão única:

- /catalogo/api/unidades/5

Este mesmo recurso, numa possível versão 2, deve ser formatado com a seguinte URI:

- /catalogo/api/v2/unidades/5

3.9. Camada de Integração

O objetivo da camada de integração é prover um meio de comunicação entre o sistema que vai ser desenvolvido e os demais sistemas que o circundam. Os objetos dessa camada fornecem à camada de negócio uma simplificação para o acesso aos sistemas externos, livrando-a da responsabilidade de tratar detalhes inerentes aos protocolos de comunicação envolvidos, formatações e conversões de tipos de dados e demais particularidades sintáticas e semânticas inerentes aos sistemas externos.

A integração com os serviços do PLE deve utilizar o Feign Client para realizar a

comunicação com estes microsserviços.

3.10. Requisitos para Implementação das Camadas

Os itens apresentados a seguir referem-se à lista de tecnologias e requisitos necessários para suportar as camadas de apresentação, negócio e persistência:

- Java 11
- Maven 3
- Docker
- NodeJS
- Undertow
- TypeScript
- Kubernetes
- Istio
- Minio
- Nginx

Bibliotecas a serem utilizadas pela aplicação:

- Spring Cloud Hoxton
- Spring Security
- Angular 11
- PrimeNG 11
- Liquibase
- Feign Client

3.11. Visão Detalhada das Camadas

A seguir, apresentamos uma visão detalhada das camadas do sistema, na qual são ilustrados os principais componentes que compõem cada camada bem como os sistemas externos com o qual o sistema deverá interagir.

3.12. Visão Geral do Processo

O diagrama de sequência abaixo apresenta o fluxo de informações do sistema e suas interfaces através das camadas arquiteturais.

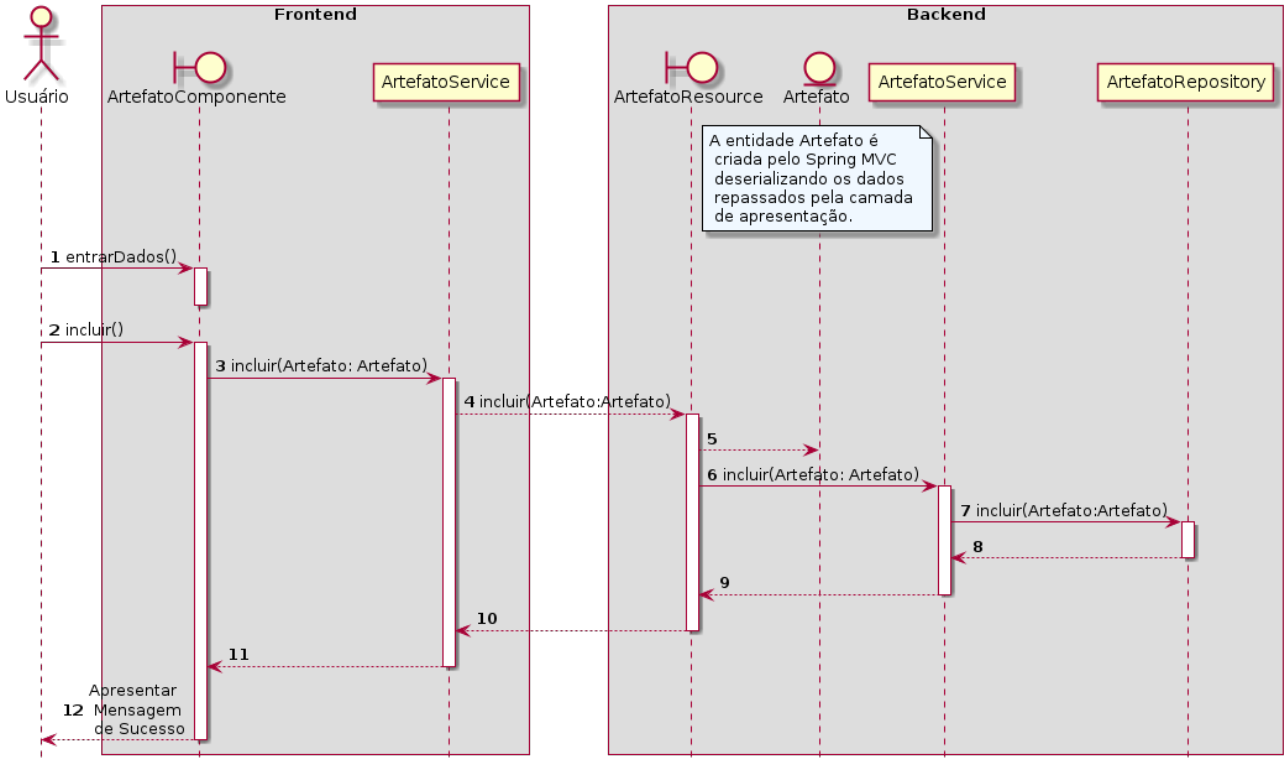

Imagem 2. Visões da arquitetura

3.13. Visão dos Módulos do Sistema

Neste item apresentamos uma visão geral dos módulos do projeto, ilustrando-os graficamente através de um diagrama. Em seguida, detalhamos as responsabilidades de cada módulo, em concordância com a especificação funcional.

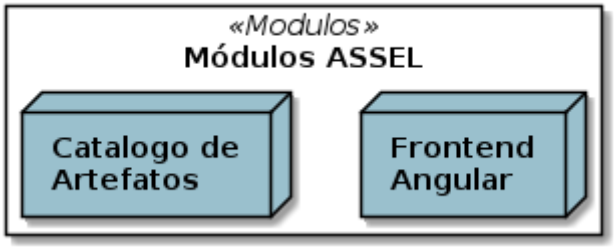

Imagem 3. Visão Geral dos Módulos

3.13.1. Front-End

Este módulo contém a aplicação client web, escrita em Angular. Ele é responsável por

fornecer as interfaces web para que os usuários interajam com os serviços executados pela aplicação.

3.13.2. Catálogo de Artefatos

Módulo responsável por realizar as regras de negócio para definir os conceitos utilizados para gerenciar os artefatos tratados pelo ASSEL. Neste módulo devem ser implementados os serviços para manter cadastro de Unidades, Ordens de Serviços e Pacotes de Artefatos que são utilizados para associar os artefatos gerenciados pela aplicação.

Além disso, este módulo é o responsável pela comunicação com o Minio para armazenar os documentos que serão catalogados pela aplicação.

Este módulo deve registrar log de auditoria para todas as operações CRUD realizadas nos serviços que ele executa, conforme for definido nos requisitos funcionais.

Este módulo também é responsável por realizar as ações de autenticação para as aplicações. Ele tem todo o fluxo de comunicação com o SSO, geração dos perfis, criação do jwt-token, cookies de segurança dentre outras ações associadas à segurança aplicada aos recursos e dados da aplicação.

3.14. Rastreabilidade

Nesta seção, é apresentada a matriz de rastreabilidade no nível de componentes arquiteturais do produto, o que permite rastrear os fontes que apoiam a realização dos requisitos propostos para o sistema alvo.

A tabela a seguir, apresenta os requisitos mapeados na proposta técnica e os respectivos componentes previstos nesta arquitetura para atendê-los.

Requisito	Front-End	Catálogo de Artefatos	Observação
Login/Logout	X	X	Seção 3.2.
Gerenciar Usuários	X	X	Seção 3.2.
Gerenciar Perfil	X	X	Seção 3.2.
Solicitar Ordem de Serviço	X	X	
Gerenciar Unidades Solicitantes	X	X	
Gerenciar Ordens de Serviço	X	X	
Gerenciar Pacotes de Artefatos	X	X	
Gerenciar Artefatos	X	X	
Log e Auditoria	X	X	
Sigilo dos Artefatos	X	X	Seção 3.2.
Exportação de dados	X	X	

3.15. Visão de Integração

A visão de integração apresenta as integrações pertinentes ao sistema. Essas integrações podem ser internas ou externas e é de suma importância que os projetistas e desenvolvedores conheçam em detalhes essas integrações, bem como os contratos e protocolos de comunicação entre os sistemas/componentes envolvidos na integração.

O Sistema deve se integrar aos seguintes sistemas a fim de atender os requisitos funcionais e não-funcionais da aplicação:

- Registro de sistemas - Istio Pilot;
- SEI-CLDF - Sistema Eletrônico de Informações;
- PLE - Processo Legislativo Eletrônico;
- SSO do CLDF.

A integração com a Servidor de configuração será feita na inicialização (bootstrap) momento em que a aplicação receberá os dados de configuração. Uma vez a inicialização terminada o serviço se registrará repassando seu identificador, endereço IP e porta de funcionamento.

O sistema irá se integrar com o SEI através de seus Web Services e com o PLE através de suas API’s REST.

3.15.1. Kubernetes

Com a escalada no uso de containers, surgiram alguns obstáculos, no sentido de como resolver a complexidade no gerenciamento de aplicações compostas por centenas de containers juntamente com o desafio de empregá-los em larga escala garantindo a elasticidade e resiliência das aplicações. Uma das soluções veio com o Kubernetes, um sistema de código aberto que foi desenvolvido pelo Google para gerenciamento de aplicativos em containers através de múltiplos hosts de cluster utilizando Docker.

Seu principal objetivo é auxiliar na adoção da tecnologia de containers pelo mercado, bem como facilitar a implantação de aplicativos baseados em microsserviços. O Kubernetes fornece uma plataforma que provê a automatização, distribuição de carga, monitoramento e orquestração entre containers, eliminando diversas ineficiências relacionadas à gestão de containers graças a sua organização em pods(menores unidades dentro de um cluster) que acrescentam uma camada de abstração aos containers agrupados.

Em resumo, o Kubernetes oferece aos usuários uma plataforma simples de gestão de infraestrutura e orquestração de aplicações com containers. Enquanto o Docker se concentra no empacotamento de uma aplicação e suas dependências num container e em sua implantação num servidor, Kubernetes vai além, sua função é orquestrar a implantação de aplicações compostas por múltiplos containers, gerenciá-las e monitorá-las, garantindo resiliência e escalabilidade em clusters de servidores distribuídos.

O uso da plataforma Kubernetes permite automatização da maior parte dos processos manuais necessários à implantação e ao dimensionamento de microsserviços, sendo utilizado como parte fundamental na infraestrutura da arquitetura, viabilizando e minimizando a complexidade da implantação de sistema distribuídos complexos.

O Rancher é um software de código aberto que possibilita a construção de uma plataforma privada para criação de containers, através dele é possível executar e gerenciar containers (Cattle e Kubernetes).

Com o Rancher, as organizações não precisam construir uma plataforma de serviços de container a partir do zero usando um conjunto distinto de tecnologias, através dele é possível administrar todos os pontos de seu ambiente, incluindo: Orquestração, balanceamento de carga, volumes e rede.

O Rancher usa o Prometheus, um kit de ferramentas open source para monitoramento e alerta, tem um bom funcionamento ao gravar séries temporais numéricas. Ele se ajusta tanto ao monitoramento centralizado em máquina (machine-centric), quanto ao monitoramento de arquiteturas dinâmicas orientadas a serviços. Em um mundo de microsserviços, fornece grande suporte as coletas e consultas de dados multidimensionais. Foi projetado para gerar confiabilidade, no intuito de ser o sistema em que se confie durante uma interrupção, no sentido de permitir e facilitar rápidos diagnósticos de problemas. Cada servidor Prometheus é independente, ou seja, não depende do armazenamento de rede ou de outros serviços remotos, a ideia é fornecer confiabilidade quando outras partes da infraestrutura estiverem quebradas, necessitando o minimo possível da infraestrutura para utiliza-lo. Os dados armazenados no Prometheus serão visualizados usando Dashboards do Grafana.

O sistema irá utilizar o Spring Cloud Kubernetes para realizar a configuração automática da aplicação com o uso ConfigMaps.

3.15.2. Istio

A adoção de microsserviços e o uso de contêineres oferecem vários benefícios para as organizações que as utilizam. No entanto, não há como negar que sua adoção pode

sobrecarregar as equipes de DevOps. Os desenvolvedores devem usar microsserviços para arquitetar a portabilidade, enquanto os operadores estão gerenciando implantações híbridas, complexas e extremamente grandes. O Istio permite conectar, proteger, controlar e observar os serviços.

Em um nível alto, o Istio ajuda a reduzir a complexidade dessas implantações e diminuir a pressão sobre as equipes de desenvolvimento. Ele é um service mesh completamente aberto que se espalha de forma transparente em aplicativos distribuídos existentes. É também uma plataforma, incluindo APIs que permitem a integração em diversas plataformas de monitoramento e controles de segurança. O conjunto diversificado de recursos do Istio permite que você execute com sucesso e eficiência uma arquitetura de microsserviço distribuída e forneça uma maneira uniforme de proteger, conectar e monitorar microsserviços, separando requisitos não funcionais de descoberta e monitoramento do código fonte dos serviços.

O termo *service mesh* é usado para descrever a rede de microsserviços e as interações entre eles. Como uma malha de serviço cresce em tamanho e complexidade, pode se tornar mais difícil de entender e gerenciar. Seus requisitos podem incluir descoberta, balanceamento de carga, recuperação de falhas, métricas e monitoramento. Uma malha de serviço também tem, frequentemente, requisitos operacionais mais complexos, como testes A/B, canary rollouts, rate limiting, controle de acesso e autenticação de ponta a ponta.

O Istio fornece insights comportamentais e controle operacional sobre a malha de serviços como um todo, oferecendo uma solução completa para satisfazer os diversos requisitos dos aplicativos de microsserviços. O uso do Istio service mesh nos microsserviços desenvolvidos em plataformas heterogêneas permitirá que eles se comuniquem de forma segura, sejam controlados e observados de forma centralizada independente de sua implantação, delegando para o service mesh essas responsabilidades e sua configuração.

A malha de serviços do Istio é logicamente divida em dois planos, o data plane (plano de dados) e o control plane (plano de controle).

Muitas vezes é importante poder visualizar a forma com a qual os microsserviços interagem e compreender as relações entre eles. Para cumprir essa tarefa o Istio dispõe de add-ons como a Kiali, uma ferramenta que disponibiliza um console para visualizar a topologia dos microsserviços, além de informações como tráfego na rede de serviços e performance, entre outras. O Kiali gera os gráficos de relacionamento entre os microsserviços a partir das traces geradas pelo Jaeger.

A aplicação irá utilizar o Istio para gerenciar a comunicação dos microsserviços por

intermédio do Kubernetes.

3.15.3. CAS

O Apereo CAS é uma ferramenta de código aberto que oferece soluções corporativas para login único, autenticação e autorização. Baseada no Spring Webflow/Spring Boot, faz uso do Spring Security e suporta diversos protocolos como SAML, WS-Federation, OAuth2, OpenID, OpenID Connect etc. Também suporta diversos mecanismos de autenticação como LDAP, JAAS, Bancos de Dados etc.

CAS é uma ferramenta concebida para total customização de telas, fluxos, modelo de dados e mecanismos de comunicação.

Através do Web Flow, o CAS permite gerenciamento do fluxo de navegação baseando-se em transições de estado. O Web Flow permite total customização através de pontos de extensão e injeção de componentes em tempo de execução. O CAS se beneficia da estratégia de WAR Overlay, de modo que a partir de uma instalação padrão é possível adicionar e subscrever comportamentos desejados.

O mecanismo Delegate Authentication permite que o CAS atue como um cliente e delegue o processo de autenticação para um provider externo de acordo com a seguinte listagem:

- CAS server
- SAML2 identity providers
- OAuth2 providers
- OpenID providers
- OpenID Connect identity providers
- ADFS

O Módulo de segurança

3.16. Visão de Classes

Nesta seção, estão descritos os diagramas/fluxogramas de classe, sequencia e atividades para os requisitos críticos do sistema.

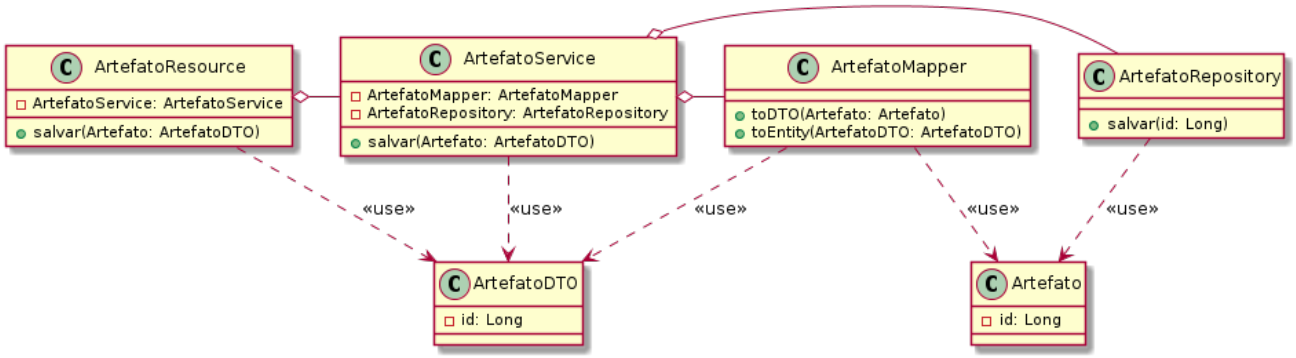

Imagem 4. Visão de Classes

4. Ambiente de Desenvolvimento

Ferramentas a serem utilizadas no ambiente de desenvolvimento:

- Eclipse IDE
- Docker
- Maven
- Java 11
- NodeJS

Ferramentas necessárias para dar suporte à aplicação e ao desenvolvimento são:

- Sql Developer

A solução prevê acessos às seguintes bases de dados:

- SQL Server 2016

5. Ambiente de Testes

Ferramentas de apoio aos testes e à codificação:

- Jenkins
- Sonar

6. Ambiente de Implantação

6.1. Ambiente de Implantação

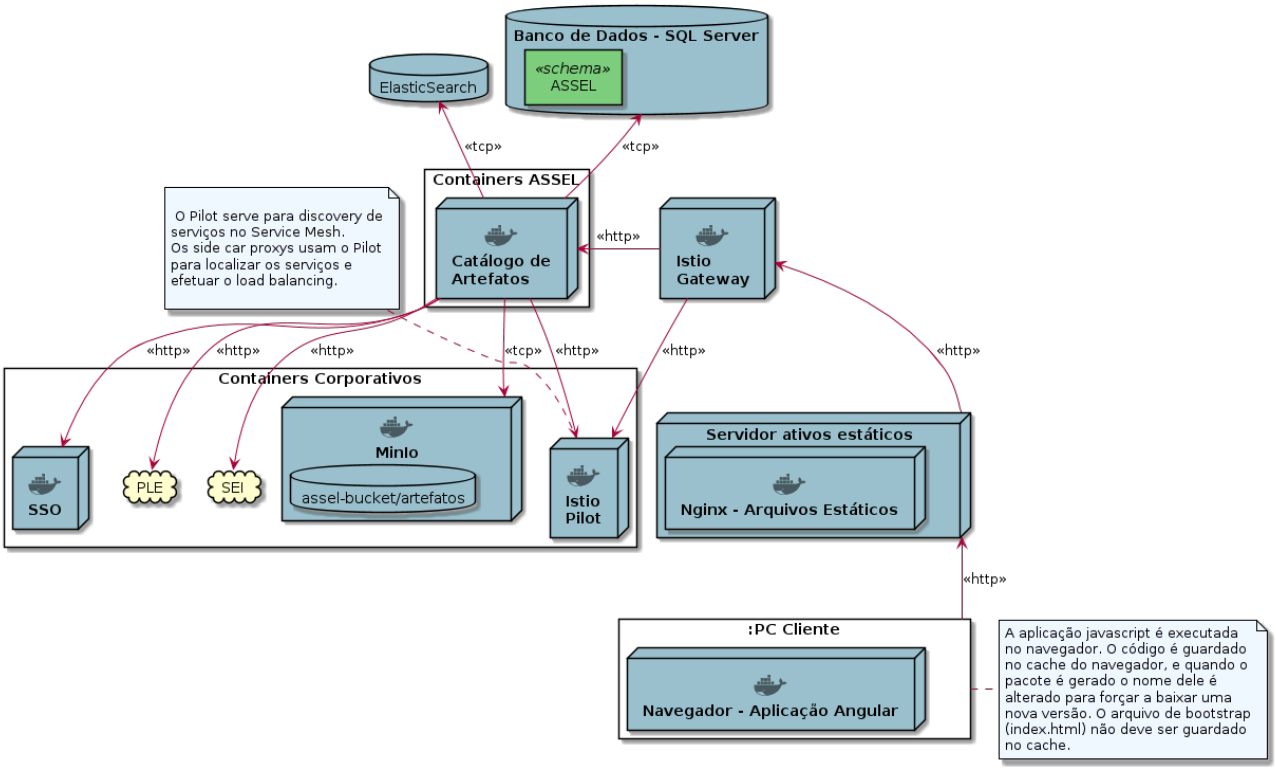

Imagem 5. Diagrama de Implantação

6.2. Componentes Principais

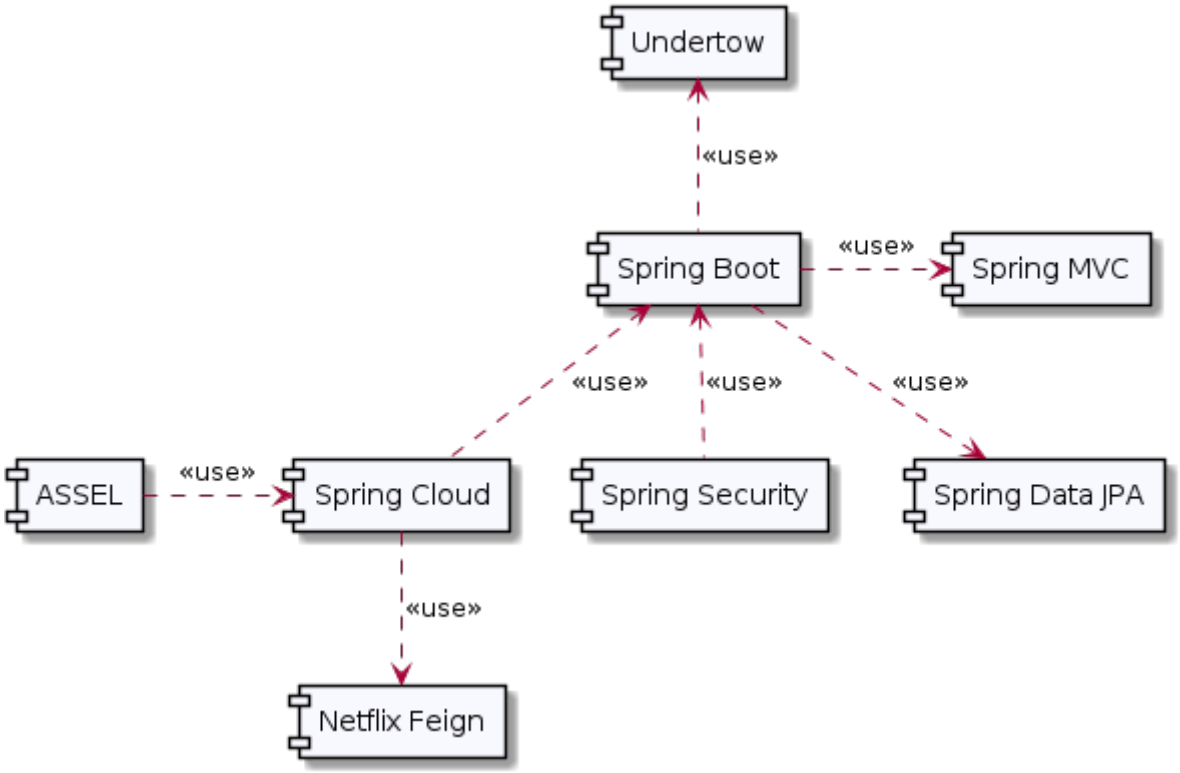

Imagem 6. Componentes Serviços