

|  |  |
| --- | --- |
| Sistema/Solução: | [nome do sistema/solução] |
| O.S. Nr: | [número da ordem de serviço] |
| Requisitante | [nome do requisitante] |

**DOCUMENTO DE ARQUITETURA DE SOFTWARE – DAS**

Sistema: [nome do sistema]

Sumário

[**1.** **Introdução** 3](#_Toc48911790)

[**2.** **Objetivo do Documento** 4](#_Toc48911791)

[**3.** **Visões da Arquitetura** 5](#_Toc48911792)

[**4.** **Ambiente de Desenvolvimento** 13](#_Toc48911793)

[**5.** **Ambiente de Testes** 14](#_Toc48911794)

[**6.** **Ambiente de Implantação** 15](#_Toc48911795)

**DOCUMENTO DE ARQUITETURA DE SOFTWARE – DAS – SIMEC (PHP)**

|  |  |  |  |
| --- | --- | --- | --- |
| Controle de Versões | | | |
| Versão | **Data** | **Autor** | **Descrição** |
| 1.0 | 17/12/2019 | BASIS | Informações extraídas do Documento de Arquitetura de Software elaborado e entregue pela fábrica de software (BASIS) em 17/12/2019. |

1. **Introdução**

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

* 1. **Definições, Acrônimos e Abreviações**

|  |  |
| --- | --- |
| Abreviação | Descrição |
| API | Application Programming Interface (Interface de Programação de Aplicativos) |
| JWT | JSON Web Tokens |
| REST | Representational State Transfer |
| SSO | Single Sign-On |
| Strangler pattern | Design Pattern para transformar de maneira incremental uma aplicação monolítica em microsserviços, substituindo uma a uma as funcionalidades por novos microsserviços |

Tabela 1-Tabela de abreviações

* 1. **Documentos de Referência**

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

[Documentação do Istio](https://istio.io/docs/)

1. **Objetivo do Documento**

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

1. **Visões da Arquitetura**
   1. **Critérios de Avaliação Arquitetural**

Não se aplica

* 1. **Arquitetura imposta pelo cliente**

Não se aplica.

* 1. **Arquiteturas descartadas**

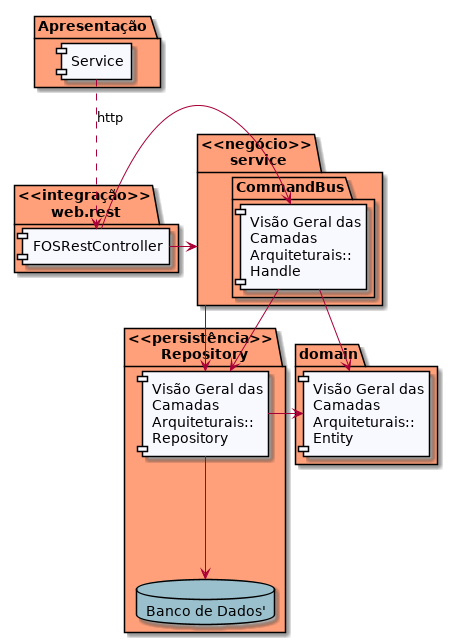
Não se aplica.

* 1. **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.



*Figure 1. Visão Geral das Camadas Arquiteturais*

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **Camada de Serviços**

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

* 1. **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.

* 1. **Requisitos para Implementação das Camadas**

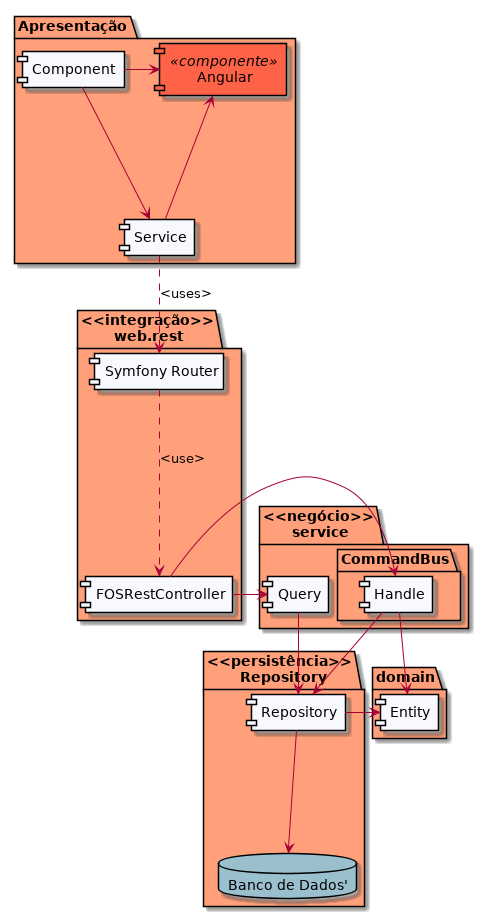
Os itens apresentados a seguir referem-se à lista de tecnologias e requisitos necessários para suportar as camadas de apresentação, negócio e persistência:

* PHP 7.2
* Docker
* NodeJS
* TypeScript
* Composer

Bibliotecas a serem utilizadas pela aplicação:

* Angular 8
* PrimeNG 8
* Symfony 5.0
* Doctrine Bundle 2.0
* League Tactician Bundle 1.1
* Friends Of Symfony - FOSRestBundle 2.6.0
* Nelmio API DOC Bundle 3.5
* Nelmio Cors Bundle 2.0
* Lexik JWT Authentication Bundle 2.6
* Symfony Swiftmailer Bundle 3.4
  1. **Visão Detalhada das Camadas**

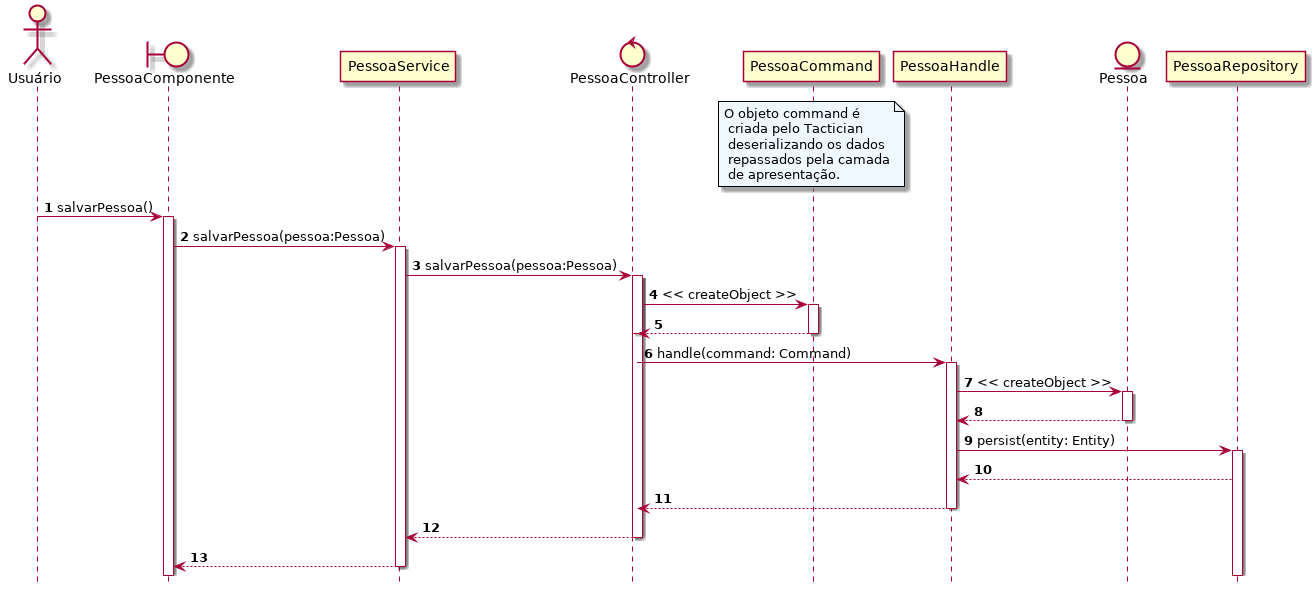
A seguir, apresentamos uma visão detalhada das camadas do sistema, na qual são ilustrados os principais componentes que compõem cada camada bem como os sistemas externos com o qual o sistema deverá interagir.



*Figure 2. Visões detalhada das camadas*

* 1. **Visão Geral do Processo**

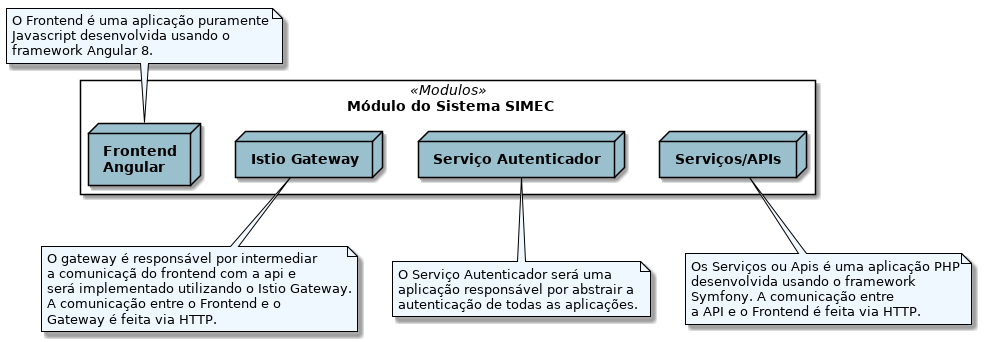
O diagrama de sequência abaixo apresenta o fluxo de informações do sistema e suas interfaces através das camadas arquiteturais.



*Figure 3. Visões da arquitetura*

* 1. **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.



*Figure 4. Módulos que compõem o sistema*

* + 1. **Módulo Frontend**

Módulo que faz interação com o cliente. Este módulo contém as regras básicas de validações de campos sem nenhuma regra de negócio que possa conter informações sensíveis do negócio. O objetivo deste módulo é coletar as informações e/ou as exibir para o usuário de forma amigável a fim de melhorar a experiência do usuário. Este módulo é organizado internamente em Componentes e Serviços.

Os componentes são responsáveis pela coleta e exibição de dados, juntamente com as opções de controles de telas. Os services são responsáveis pela validação dos dados recebidos pelos componentes e além disso fazer a comunicação com a API.

* + 1. **Módulo Gateway**

Módulos responsáveis por encaminhar as requisições para os serviços. O Gateway também adicionará no cabeçalho da requisição o token que será recuperado do cookie de sessão. Esse procedimento possibilita a integração dos novos serviços com os sistemas legados de forma transparente, permitindo assim uma maior flexibilidade na organização da infraestrutura e a implementação dos serviços. A implementação será feita utilizando o Istio Gateway.

* + 1. **Módulo Serviço Autenticador**

Módulos responsáveis por realizar a autenticação dos serviços no SSO e disponibilizar uma interface estável entre os serviços e os clientes, permitindo assim uma customização da lógica de autenticação de forma centralizada.

* + 1. **Módulos Serviço/API**

Módulos responsáveis por realizar validações de regras de negócio, persistência de dados e comunicação com sistemas externos. Este módulo é organizado em Resources Rest, Services, Repositories e Entities.

* Os Resources Rest são o ponto de entrada da comunicação com o backend. Eles fazem apenas a validação de obrigatoriedade dos campos. Os resources recebem como entrada objetos que são gerados automaticamente pelo framework Symfony fazendo o parse dos JSONs recebidos no corpo dos requests.
* Os Services são os detentores das regras de negócio. Eles recebem os dados dos Resources Rest e fazem todas as validações de negócio. Dentro dos services, são criadas as entidades de persistência que serão repassadas para os repositórios de dados. Essas entidades de persistências, não são repassadas à camada de resources rest para garantir um desacoplamento entre as camadas de controle, negócio e persistência.
* Os Repositories são responsáveis por fazer a persistência dos dados com a base de dados. Eles fazem apenas o envio, atualização e recuperação de dados.

* 1. **Rastreabilidade**

Não se aplica

* 1. **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.

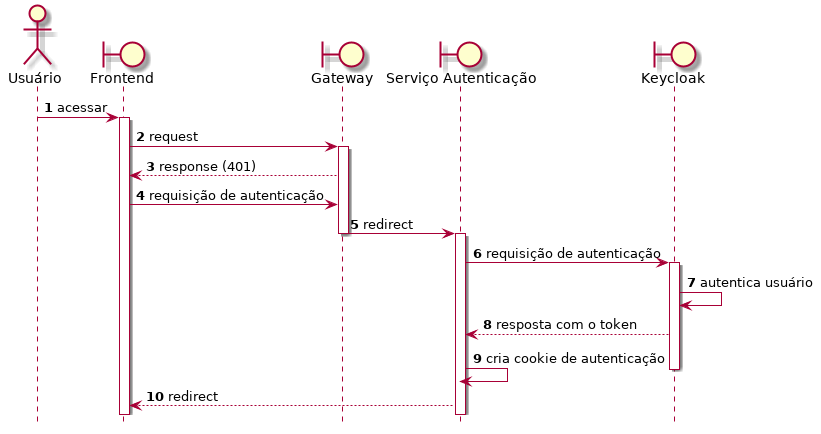
A fim de atender os requisitos funcionais e não-funcionais da aplicação, o sistema deve se integrar aos seguintes serviços:

* Outros Serviços/APIS
* Istio Gateway
* Serviço Autenticador
* Keycloak
  + 1. **Integração com outros Serviços/APIS**

A integração com outros serviços deverá ser feita utilizando o padrão REST (Representational State Transfer). Nesse modelo, a "Aplicação A" deverá realizar uma requisição HTTP a "Aplicação B" a fim de obter algum tipo de informação que é do seu domínio. A "Aplicação B" resolverá a requisição e devolverá uma resposta para a "Aplicação A". A aplicações estarão protegias atravez da lógica de autenticação por token JWT.

* + 1. **Integração com o Keycloak**

O diagrama abaixo apresenta o fluxo de integração de uma aplicação e o Keycloak.



*Figure 5. Visões da integração com o Keycloak*

A integração com o Keycloak será utilizada para autenticação dos usuários do sistema através do protocolo da autenticação Open ID Connect (OIDC).

* + 1. **Integração com módulos antigos**

A integração com módulos antigos será feita com o uso do Strangler Pattern por intermédio do Gateway, este também fará o papel de autenticação para os serviços antigos. Para que seja possível a comunicação proposta, o SIMEC legado deverá ser adicionado a estrutura do cluster e colocado sob a proteção do Gateway do Istio, possibilitando assim uma migração incremental, com a menor intervenção possível nos sistemas legados.

A comunicação entre os serviços dos módulos novos e dos módulos antigos será feita utilizando o padrão REST. Os dados não serão acessados por meio do banco de dados, caso haja necessidade de acesso aos dados legados, serão criados novos microsserviços para acessarem as bases legadas.

* 1. **Visão de Classes**

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

N/A

1. **Ambiente de Desenvolvimento**

Ferramentas a serem utilizadas no ambiente de desenvolvimento:

* Eclipse IDE
* Docker
* Composer
* PHP 7.2
* NodeJS
* GIT

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

* DBeaver

A solução prevê acessos às seguintes bases de dados:

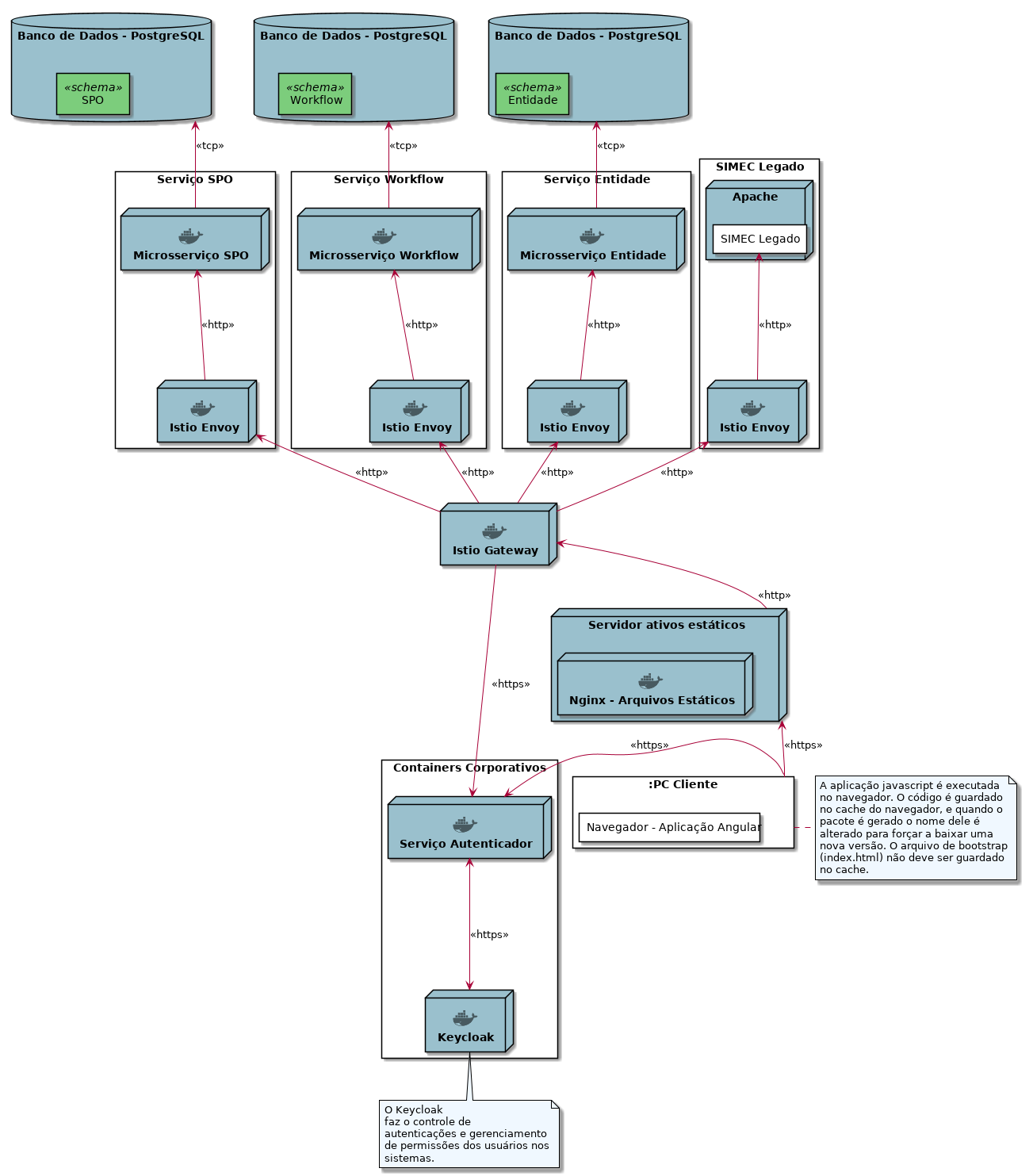
* Mysql
* Microsoft SQL Server
* Oracle
* PostgresSql

1. **Ambiente de Testes**

Ferramentas de apoio aos testes e à codificação:

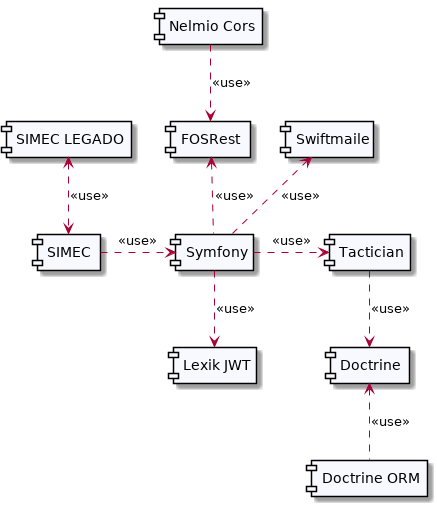
* Jenkins
* Sonar

1. **Ambiente de Implantação**
   1. **Ambiente de Implantação**



*Figure 6. Diagrama de Impantação*

* 1. **Componentes Principais**



*Figure 7. Componentes Serviços*