



INEP - PHP

Documento de Arquitetura de Software

Enus Nascimento

Versão 1.0.0, 18/08/2023

Índice

1. Introdução	2
1.1. Definições, Acrônimos e Abreviações	2
1.2. Documentos de Referência	2
2. Objetivo do Documento	2
3. Visões da Arquitetura	2
3.1. Critérios de Avaliação Arquitetural	2
3.2. Arquitetura imposta pelo cliente	3
3.3. Arquiteturas descartadas	3
3.4. Visão Geral das Camadas Arquiteturais	3
3.5. Camada de Apresentação	4
3.6. Camada de Controle	4
3.7. Camada de Negócio	5
3.8. Camada de Persistência	5
3.9. Camada de Integração	5
3.10. Requisitos para Implementação das Camadas	6
3.11. Visão Geral do Processo	6
3.12. Visão dos Módulos do Sistema	7
3.13. Rastreabilidade	7
3.14. Visão de Integração	7
3.15. Visão de Classes	9
4. Ambiente de Desenvolvimento	9
5. Ambiente de Testes	10
6. Ambiente de Implantação	10
6.1. Diagrama de Implantação	10
6.2. Componentes Principais	11

Histórico de Revisões

Data	Versão	Descrição	Autor	Revisor
18/08/2023	1.0.0	Criação do documento	Enus Nascimento	Vitor Ferreira

1. Introdução

Este documento visa descrever a arquitetura lógica e física para projetos desenvolvidos utilizando a linguagem PHP.

1.1. Definições, Acrônimos e Abreviações

- **API** (Application Programming Interface): É qualquer interface que expõe as funcionalidades de um sistema.
- **REST** (Representational State Transfer): É um modelo de arquitetura que descreve regras para expor serviços através de recursos e determinar ações sobre esses recursos de acordo com os verbos HTTP.
- **JSON** (JavaScript Object Notation): é uma formatação leve de troca de dados.
- **JWT** (JSON Web Token): Padrão utilizado para realizar autenticação entre duas partes por meio de um token assinado.
- **CRSF** (Cross-site request forgery): Ataque que ocorre quando uma requisição HTTP é feita entre sites na tentativa de se passar por um usuário legítimo.
- **SemVer** Especificação de Versionamento Semântico. *

1.2. Documentos de Referência

- [Laravel Framework](#)
- [SemVer](#)

2. Objetivo do Documento

O objetivo deste documento é apresentar ao cliente a descrição da arquitetura para projetos PHP.

3. Visões da Arquitetura

3.1. Critérios de Avaliação Arquitetural

N/A

3.2. Arquitetura imposta pelo cliente

O cliente solicitou a utilização de uma arquitetura PHP utilizando o framework Laravel no back-end. A arquitetura do sistema Syspag foi utilizada como modelo para as definições arquiteturais presentes neste documento.

3.3. Arquiteturas descartadas

N/A

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.

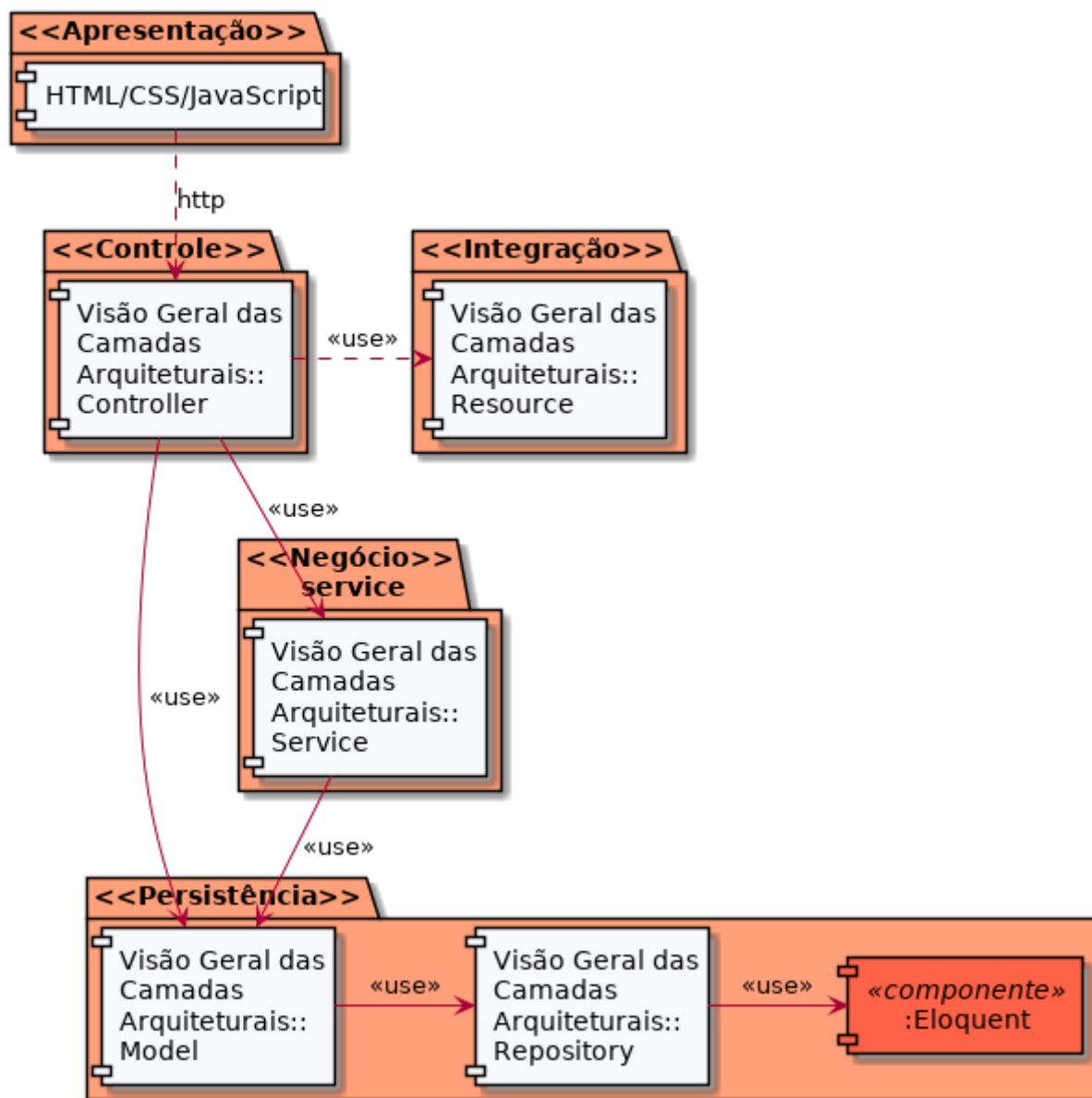


Imagem 1. Visão Geral das Camadas Arquiteturais

3.5. Camada de Apresentação

A camada de apresentação é responsável por fazer a lógica de construção das páginas para serem exibidas para os usuários, tratar os eventos do browser, como cliques, e gerenciar o fluxo de execução do sistema. Essa camada pode ser construída utilizando o frameworks de front-end como **Angular**. Toda a estrutura deve ser mantida como um projeto independente. A comunicação com o back-end deverá ocorrer pelo consumo de uma API utilizando o estilo arquitetural REST.

3.6. Camada de Controle

O objetivo da camada de controle é prover um meio de comunicação entre a camada de

apresentação e as demais camadas do sistema que vai ser desenvolvido. Essa camada será basicamente uma API REST.

3.7. 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.

3.8. 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 de dados e o problema do mundo real.

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 controle 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.

3.9.1. GraphQL

GraphQL é uma linguagem de consulta para APIs em tempo de execução para atender essas consultas com seus dados existentes. O GraphQL fornece uma descrição completa e compreensível dos dados em sua API

GraphQL é uma linguagem de consulta para APIs em tempo de execução para atender a essas consultas com seus dados existentes. O GraphQL fornece uma descrição completa e compreensível dos dados em sua API, dá aos clientes o poder de solicitar exatamente o que precisam e nada mais, facilita a evolução das APIs ao longo do tempo e permite ferramentas poderosas para desenvolvedores.

- Os arquivos graphql referente ao schema, query, type, deve ter extensão **.graphql**

- Os type devem ficar no diretório **routes/graphql/**. No schema principal (**routes/graphql/schame.graphql**) deve ser feito a importação do arquivo.

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 do **back-end** (Controle, Negócio, Persistência e Integração):

- PHP 7.1
- Laravel Framework 5.7
- Composer
- BCMath PHP Extension
- Ctype PHP Extension
- Fileinfo PHP Extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- OCI8 Extension
- inep/infrastructure-laravel 1.0,

3.10.1. Requisitos para persistência de Banco de Dados

- Oracle.

3.11. Visão Geral do Processo

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

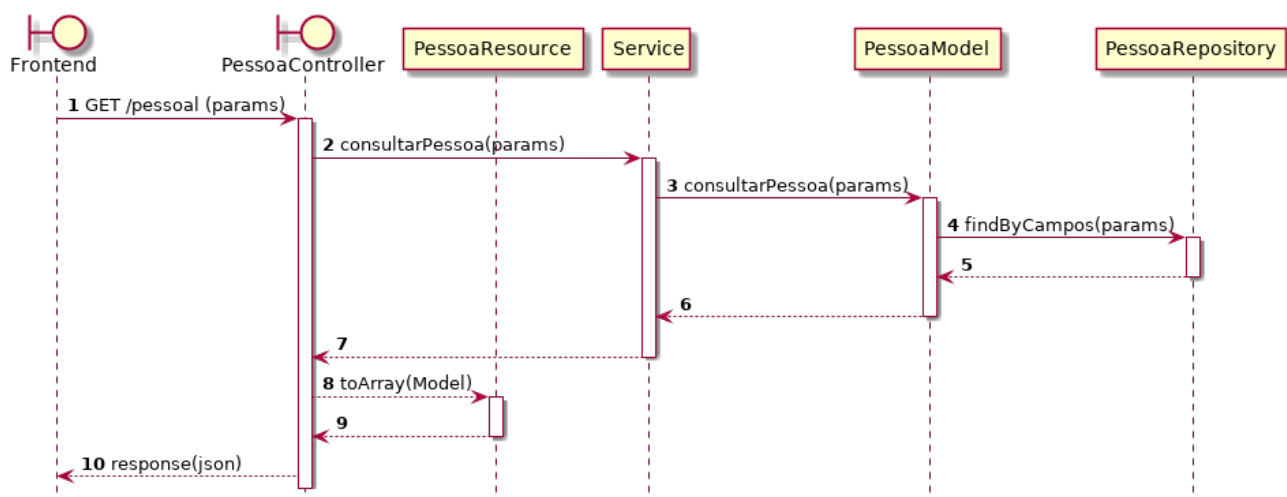


Imagem 2. Visões da arquitetura

3.12. 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.

Para o sistema serão criados os seguintes módulos:

- **Principal:** contém as funcionalidades gerais do sistema.

3.13. Rastreabilidade

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

Origem do Requisito	Item no DAS
OS: Definição das tecnologias	3.10. Requisitos para Implementação das Camadas

3.14. Visão de Integração

A visão de integração apresenta as integrações pertinentes ao projeto. 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 integrar-se aos seguintes sistemas a fim de atender os requisitos

funcionais e não-funcionais da aplicação:

3.14.1. OpenID

OpenID é uma camada de autenticação que opera sobre o protocolo OAuth2 e permite verificar a identidade de um usuário já autenticado em um servidor de autorização.

Abaixo, o exemplo de um fluxo padrão via OpenID:

- O usuário precisa acessar sua conta dentro do sistema.
- O sistema solicita ao usuário o OpenID.
- O Usuário entra com o OpenID.
- O sistema redireciona o usuário para o provider do OpenID (nesse caso o Acesso GovBR).
- O usuário autentica no provedor do OpenID.
- O provedor do OpenID redireciona o usuário de volta ao sistema.
- O sistema verifica e confirma a autorização do usuário.

OpenID utiliza o chamado `id_token`, que é um token de segurança que permite ao cliente verificar a identidade do usuário e obter as informações básicas do seu perfil.

Para o controle de autorização é utilizado o protocolo OAuth2, entre outras coisas, permite aos clientes obterem acesso a recursos protegidos do lado do servidor em nome do proprietário desses recursos. A concessão da autorização dos recursos pelo seu proprietário é representada por uma credencial, que no caso do Brasil Cidadão é um código de autorização baseado em tokens JWT.

Um token JWT é dividido em 3 partes:

- Header: São objetos JSON definidos por 2 atributos. O tipo do token (`typ`) que é o JWT e o algoritmo (`alg`) de encriptação utilizado, como HMAC, SHA256 ou RSA.
- Payload: São todos ou parte dos atributos de uma entidade representadas em JSON:
- Signature: Para criar a assinatura temos que assinar o header codificado, o payload codificado e informar o `secret`, que no caso abaixo é uma palavra secreta definida na aplicação. A assinatura é criada para verificar se quem enviou a requisição é quem realmente diz ser.

3.15. Visão de Classes

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

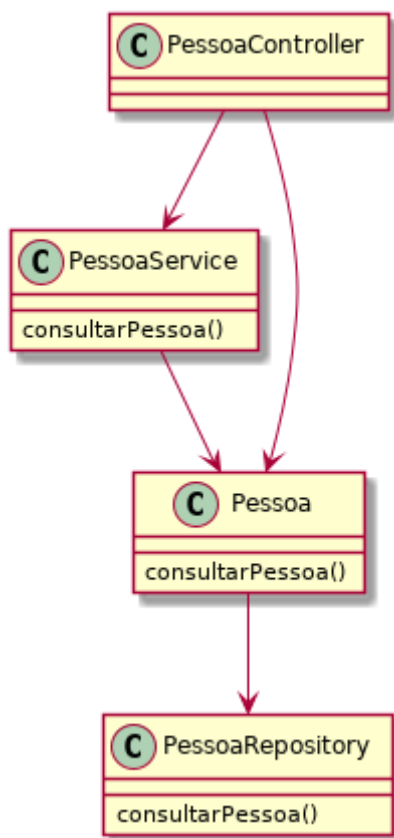


Imagem 3. Diagrama de classes

4. Ambiente de Desenvolvimento

Ferramentas a serem utilizadas no ambiente de desenvolvimento:

- Docker
- Composer
- PHP 7.1

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

- DBeaver
- Ferramenta de modelagem de banco de dados

5. Ambiente de Testes

As seguintes ferramentas serão utilizadas no ambiente de teste:

- Jenkins
- Sonar 8.4.2 ou superior
- Sonar Scanner
- PHP Unit

6. Ambiente de Implantação

6.1. Diagrama de Implantação

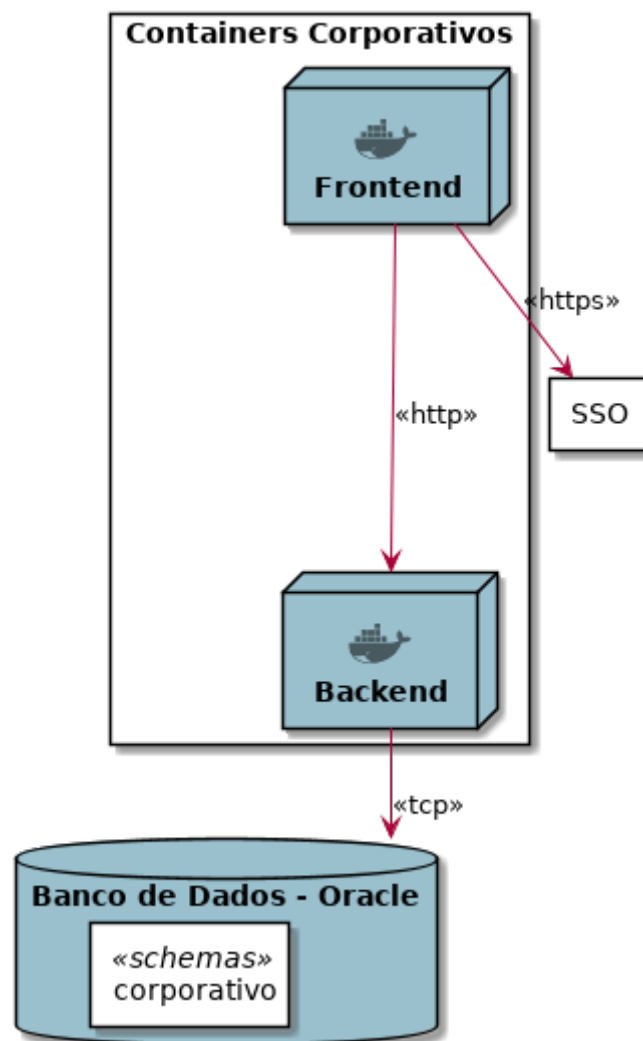


Imagem 4. Diagrama de Implantação

6.2. Componentes Principais

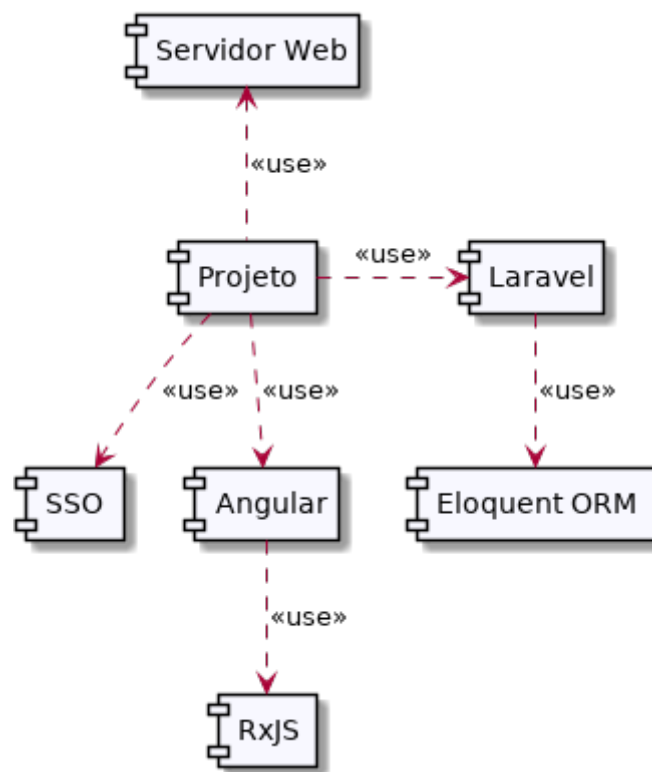


Imagem 5. Componentes Serviços