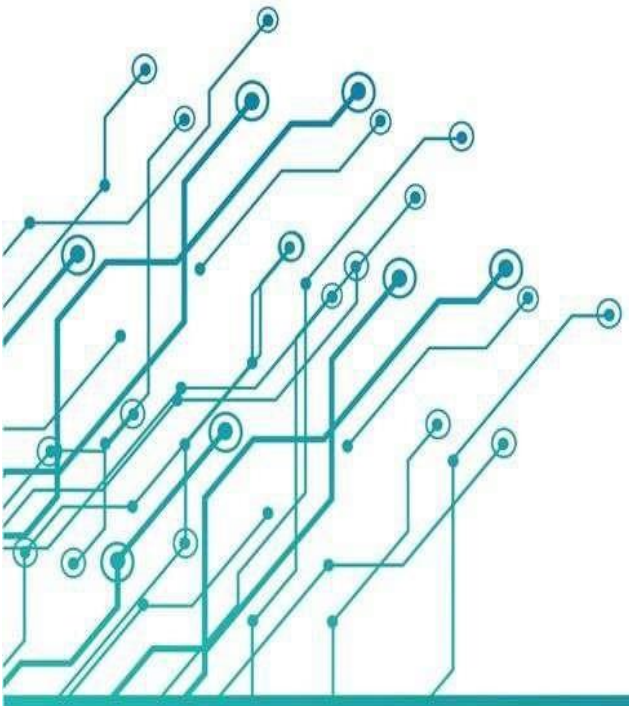
STIC Subsecretaria de Tecnologia da Informação e Comunicação

Documento de Arquitetura PHP 3.0



MINISTÉRIO DA EDUCAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Sumário

DOCUMENTO DE ARQUITETURA PHP 3.0

Controle de Versões			
Versão	**Data**	**Autor**	**Descrição**
1.0	18/10/2021	Wisley Alves couto	Criação do documento
1.1	18/10/2022	JOSÉ CARLOS FERNANDES	Atualização do documento
1.2	09/03/2023	Wisley Alves couto	Atualização do documento
1.3	14/05/2024	Wenderson Alves de Aguiar	Atualização do documento

Tabela 1-Tabela de abreviações

1. Introdução

Este documento visa descrever a arquitetura lógica e física dos projetos na linguagem de programação PHP a serem desenvolvidos e mantidos no ambiente do Ministério da Educação.

Este documento se aplica somente para os sistemas novos, ou seja, aqueles que ainda serão desenvolvidos e entregues ao MEC.

1.1. Definições, Acrônimos e Abreviações

Abreviação	Descrição
API	Application Programming Interface (Interface de Programação de Aplicativos) Uma API é um conjunto de definições e protocolos que permite que diferentes softwares se comuniquem entre si. Ela funciona como um intermediário, facilitando a troca de dados e funcionalidades entre aplicativos, sem a necessidade de entender os detalhes internos de cada um.
JWT	JSON Web Tokens (Chave de Conexão para Objetos Secretos) é um padrão aberto para a autenticação segura e transmissão de informações entre duas partes. Ele se tornou uma ferramenta essencial para APIs e aplicações modernas, devido à sua simplicidade, flexibilidade e segurança.
REST	Representational State Transfer (Formato de Transferência de Dados pela Web) se trata de um estilo arquitetônico para interfaces de software na web, definido por um conjunto de princípios e regras. Ele foi criado por Roy Fielding em 2000 e se tornou a base para muitas Interfaces de Programação de Aplicações modernas.

SSO	Single Sign-On (Login único) é uma tecnologia que permite que os usuários acessem vários aplicativos e serviços usando apenas uma única autenticação. Como exemplo: acessar sua conta bancária, e-mail, rede social e sistema interno da empresa com apenas uma senha.
STRANGLE PATTERN	Strangler Pattern (Padrão Estrangulador) é uma ação adotada para transformar de maneira incremental uma aplicação monolítica em microsserviços, substituindo gradualmente sistemas legados para arquiteturas mais modernas, como microsserviços.
HELM	Helm Package Manager (gerenciador de pacotes Helm), facilita a instalação, o gerenciamento e a atualização de aplicações em clusters Kubernetes. Usa um único comando para instalar e configurar um software complexo no seu cluster, sem precisar se preocupar com detalhes de manifestos e configurações.
CHART	Chart Package (Pacote Chart), é um gerenciador para aplicação no mundo do Kubernetes, um Chart é como um manual de instruções detalhado para instalar e configurar uma aplicação complexa de forma rápida e eficiente. Tudo para colocar seu software em funcionamento em um único pacote.
GITLAB	O GitLab é um gerenciador de repositório de software baseado em git que oferece um conjunto abrangente de ferramentas para gerenciar todo o ciclo de vida do desenvolvimento de software, desde o planejamento e codificação até a entrega e monitoramento. Todo o gerenciamento dos seus projetos de software em um único lugar.
MIGRATION	As migrações são essenciais para gerenciar as alterações na estrutura do seu banco de dados de forma organizada e segura. Criando o histórico completo das mudanças realizadas, permitindo reverter para versões anteriores caso necessário, criando um "versionamento" para seu banco de dados.
Container	É um ambiente isolado com uma versão de código-fonte autossuficiente que empacota uma aplicação com todas as suas dependências necessárias para ser executada: código, bibliotecas, sistema operacional (SO) e configurações. Como se fosse uma caixa isolada que guarda tudo que o software precisa para funcionar, independentemente do ambiente físico onde ele esteja rodando.

2. Objetivo do Documento

O objetivo deste documento é apresentar a padronização para os futuros sistemas que serão desenvolvidos por empresas parceiras do MEC, sejam elas universidades, institutos federais ou empresas regidas pelos contratos administrativos.

3. Visões da Arquitetura

3.1. 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.

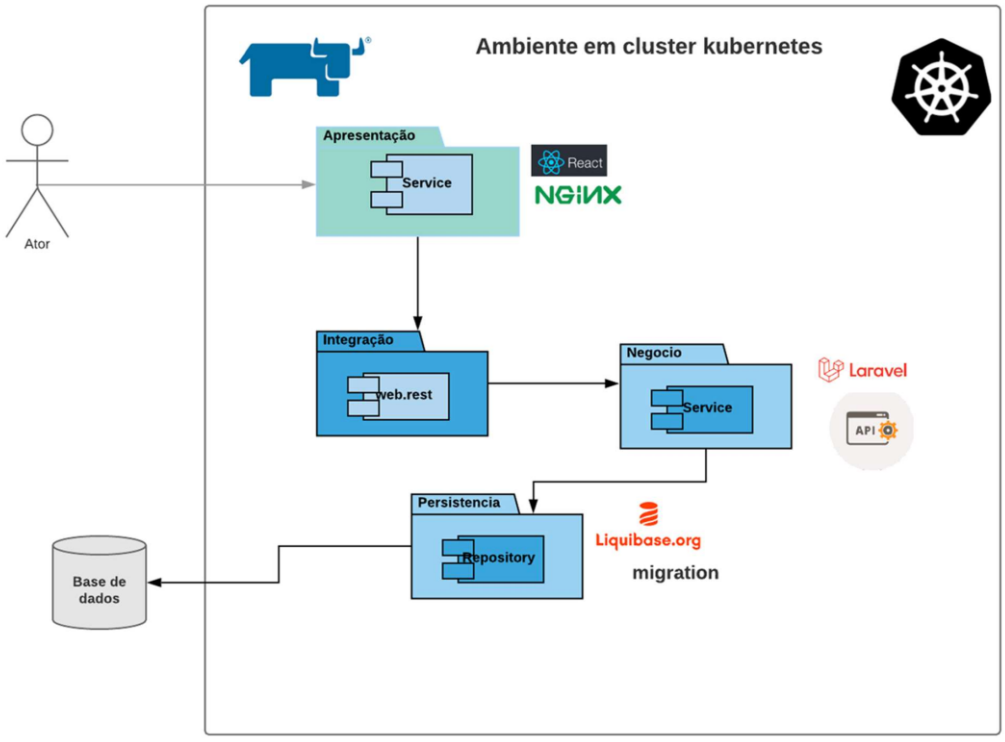


Figure 1. Visão Geral das Camadas Arquiteturais

4. 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.

4.1. 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.

4.2. Camada de Negócio

A camada de negócio 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.

4.3. 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.

4.4. Camada de Serviços

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

5. Ambiente de Desenvolvimento e Ambiente em Contêiner

Todas as aplicações a serem desenvolvidas e entregue ao MEC deverão prever arquitetura híbrida com backend separado do frontend, conforme imagem abaixo:

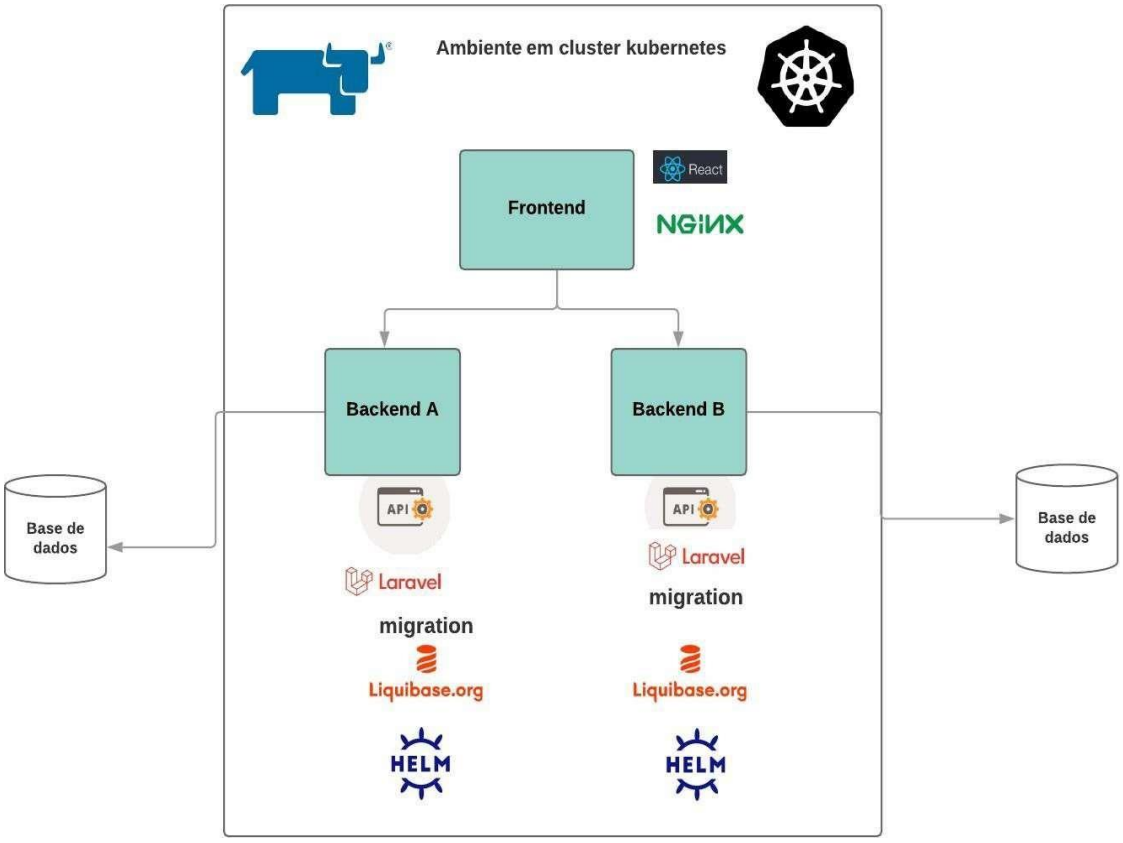


Figure 2. Visão arquitetura hibrida

Arquitetura hibrida é aquela onde se tem bem definido, uma camada de backend e outra de frontend com distribuição de responsabilidade, o front-end está diretamente exposto ao usuário final, como resultado, esses aplicativos geralmente são sensíveis ao desempenho e podem estar sujeitos a frequentes mudanças de versões. À medida que novos recursos e melhorias são desenvolvidos e incorporados ao backend os resultados são visíveis aos usuários finais. A figura 03 representa arquitetura usada pelo MEC. Como resultado, atualmente todas as aplicações deverão ser desenvolvidas nessa estrutura no qual é requisito obrigatório no momento da implantação no ambiente DevOps do MEC.

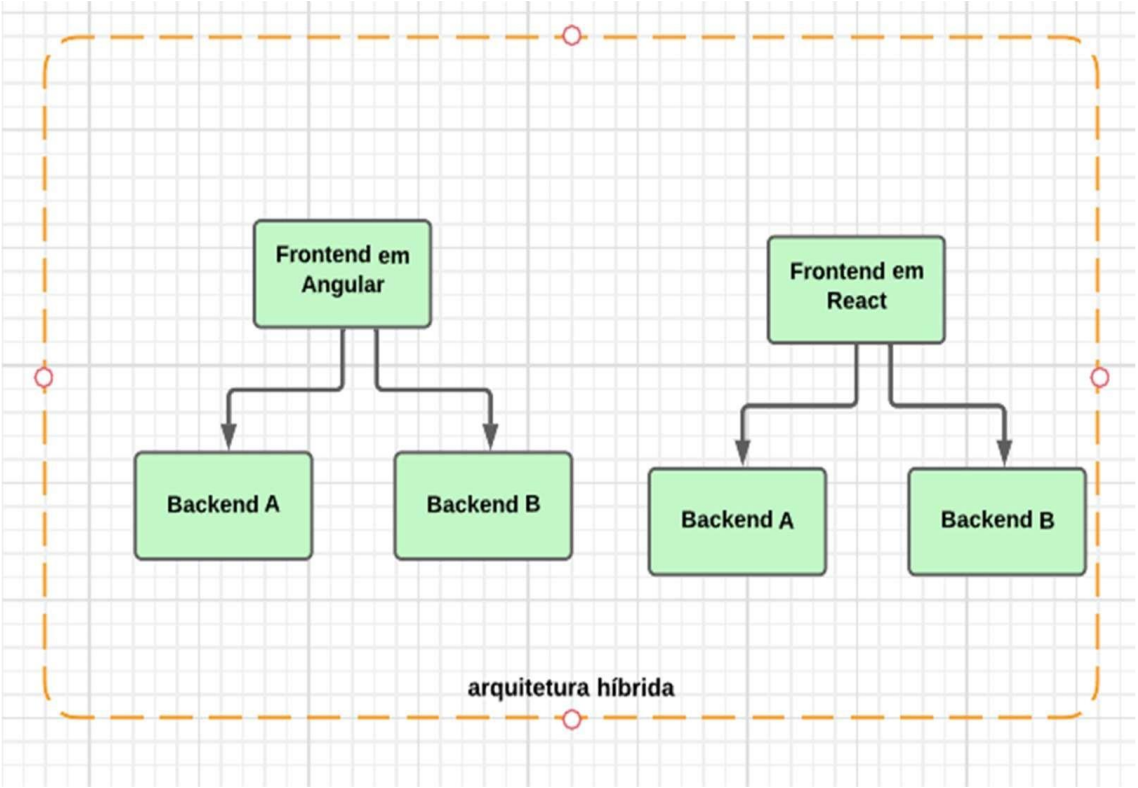


Figura 03 - Imagem arquitetura hibrida

5.1. 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.

5.2. 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.

5.3. 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.

5.4. 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 através da lógica de autenticação por token JWT. Os dados a serem tratados nessa integração devem obedecer ao tipo JSON.

6. Inicialização do Sistema/Projeto

Ao ser iniciado um projeto no ambiente do MEC para as aplicações que serão desenvolvidas na linguagem PHP será disponibilizado pela equipe de arquitetura da CGS um esqueleto denominado modeloapp, disponível na url abaixo, no qual se prevê os arquivos e arquitetura inicial para o início do desenvolvimento.

Link do Gitlab do Backend: https://gitlabbuilder.mec.gov.br/templates-dev/template-backend-php-devopsv2 Link do Gitlab do Charts: https://gitlabbuilder.mec.gov.br/templates-dev/template-charts-devops-v2

Linkd do gitlab do template docker: https://gitlabbuilder.mec.gov.br/templates-dev/template-docker-v2

Link do Gitlab do Frontend: https://gitlabbuilder.mec.gov.br/templates-dev/template-frontend-reactjs- devopsv2

Frontend deverá ser desenvolvido em conjunto com o ReactJs e fazendo uso da interface visual do Design System do Governo Federal (DSGOV) disponível em https://dsgov.estaleiro.serpro.gov.br/

Aplicação também deverá contemplar autenticação com acesso Gov o roteiro de integração está disponível na Url: https://manual-roteiro-integracao-login-unico.servicos.gov.br/pt/stable/iniciarintegracao.html

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

Todos as aplicações deverão prever a integração com acesso Gov, que garante a identificação de cada cidadão que acessa os serviços digitais do governo, os meios para realizar essa integração, tais como cadastro de client ID serão fornecidos pela área gestora da coordenação geral de sistema do MEC.

6.1. Ferramentas recomendadas no ambiente de desenvolvimento

- Docker
- Docker-compose
- Composer (gerenciador de dependências)
- PHP 8.2
- Laravel
- GITLAB
- Rancher
- Helm
- Chart

6.2. Banco de Dados

A solução pode prever acessos às seguintes bases de dados com as respectivas versões:

- Mysql versão (5.5.61);
- Microsoft SQL Server versão (SQL Server 2017 - 14.0.3281.6 – Enterprise Edition (64-bit));
- Oracle versão (12.2.0.1);
- Oracle Exata on-premise;
- PostgresSql versão (11.00);
- Mongo versão 6.0.3

Em relação aos SGBDS que serão adotados para aplicação a ser construída, deverá ser levado em consideração a necessidade da aplicação e será uma decisão em conjunto, envolvendo equipe técnica do MEC e equipe técnica da contratada.

7. Qualidade de código e auditoria nas entregas

O ambiente de integração contínua prevê testes com uso da ferramenta sonarqube, gitlab (CodeSniffer e Mess Detector) com a finalidade de analisar os padrões de código, auxiliando na qualidade do código produzido. O codeSniffer realizará auditoria no repositório do gitlab da aplicação no qual será verificado as melhores práticas do mercado para o desenvolvimento de software utilizando, no caso do backend, PSR (PHP Standard Recommendation). A equipe de arquitetura do MEC poderá no momento de a entrega realizar auditório de código-fonte, auditoria realizada, será de forma manual no qual o código-fonte será analisado e ser for necessário solicitar os ajustes a contratada, desse modo a mesma será notificada para realizar os devidos ajustes, com o objetivo de que entrega esteja nos padrões de qualidade exigidos pelo MEC.

8. Ambiente de Homologação

O ambiente de homologação será usado para que o gestor do Aplicação realize os devidos testes de aceitação na aplicação que está sendo desenvolvida ou correção de algum bug, o ambiente de homologação é de responsabilidade da equipe de arquitetura e dos gestores do MEC. Após o aceite formal por parte da área negocial, a aplicação estará apta para ser disponibilizada em produção.

9. Ambiente de Implantação do Rancher

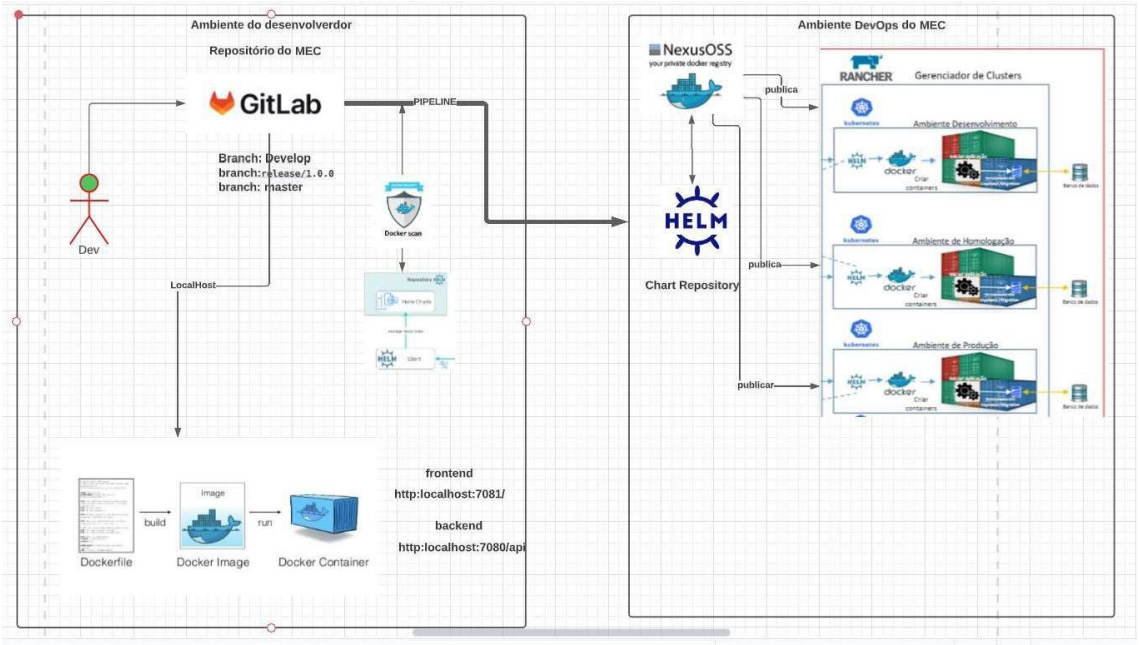


Figure 4. Diagrama de Implantação

Branch Master: Essa branch é de total responsabilidade da equipe de INFRAESTRUTURA do MEC, visto que a integração-continua escuta essa branch e qualquer atualização nesta, a pipeline de deploy é disparada e assim o ambiente de produção é atualizado de forma automática, não havendo intervenção humana.

Desta forma os deploys em produção partirão obrigatoriamente do ambiente de homologação e serão disparados para equipe de infra via chamado (BMC). Caso tenha script de banco de dados, antes da execução do script um chamado deverá ser encaminhado para equipe de administração de dados para validação. O chamado deverá ser criado pelo responsável designado do sistema, e com o apoio das informações repassadas pelo time de arquitetura.

Branch Homologação: Essa branch é de total responsabilidade da equipe de ARQUITETURA do MEC, visto que a integração continua escuta essa branch e qualquer atualização nesta, a pipeline de deploy é disparada e assim o ambiente de homologação é atualizado de forma automática, não havendo intervenção humana.

Branch Desenvolvimento: Essa branch é de total responsabilidade da equipe da FABRICA DE SOFTWARE do MEC, visto que a integração continua escuta essa branch e qualquer atualização nesta, a pipeline de deploy é disparada e assim o ambiente de desenvolvimento é atualizado de forma automática, não havendo intervenção humana.

9.1. Estrutura dos ambientes no Gitlab

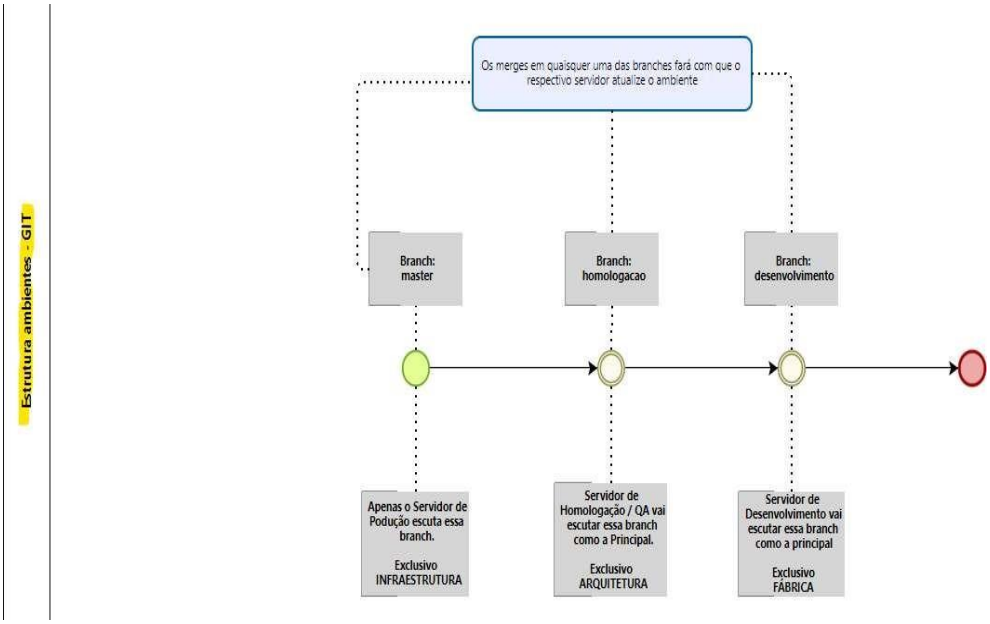


Figure 5. Estruturas de branch do gitlab

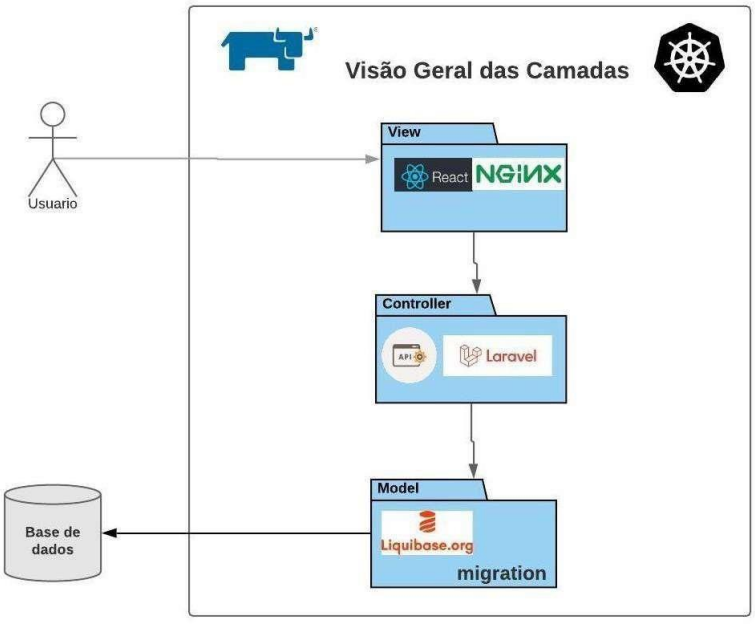


Figure 6. Visão geral das camadas

View: é responsável pela interface que será apresentada, mostrando as informações do Model para o usuário

Controller: É a camada de controle, responsável por ligar o Model e a View, fazendo com que os Models possam ser repassados para as Views e vice-versa

Model: é responsabilidade dos Models é representar o negócio. Também é responsável pelo acesso e manipulação dos dados na sua aplicação

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:

- PHP 8.2
- Composer
- Docker
- Laravel
- Symfony
- ReactJs

10.1. Bibliotecas a serem utilizadas pela aplicação:

- Dsgov
- @govbr/dsgov": "^2.0.2,
- @reduxjs/toolkit": "^1.6.0,
- "bootstrap": "^4.6.0,
- "react": "^18.2.0",
- "typescript": "^5.3.3",
- react-dom": "^18.2.0,
- react-flatpickr": "^3.10.13",
- react-redux": "^7.2.4",
- vite": "^5.1.4"
- React-dsgov-mec;

Aplicação construída por parceiros (Universidades e Institutos) ou por fábrica de desenvolvimento deverá obrigatoriamente contemplar design system do governo federal o roteiro de integração está disponível nas Urls: https://designsystemsbrasileiros.com/dsgov e https://www.gov.br/ds/home

11. Estrutura do Laravel

O Laravel se baseia no padrão MVC (Modelo-Visão-Controlador), que separa as responsabilidades da aplicação em três camadas distintas:

- Modelo (Model): Responsável por gerenciar dados, acessando e manipulando informações do banco de dados.
- Visão (View): Encarregada da interface do usuário, exibindo os dados de forma amigável para o usuário final.
- Controlador (Controller): Atua como o maestro da aplicação, recebendo requisições do usuário, processando-as e interagindo com os Models e Views para gerar a resposta adequada.

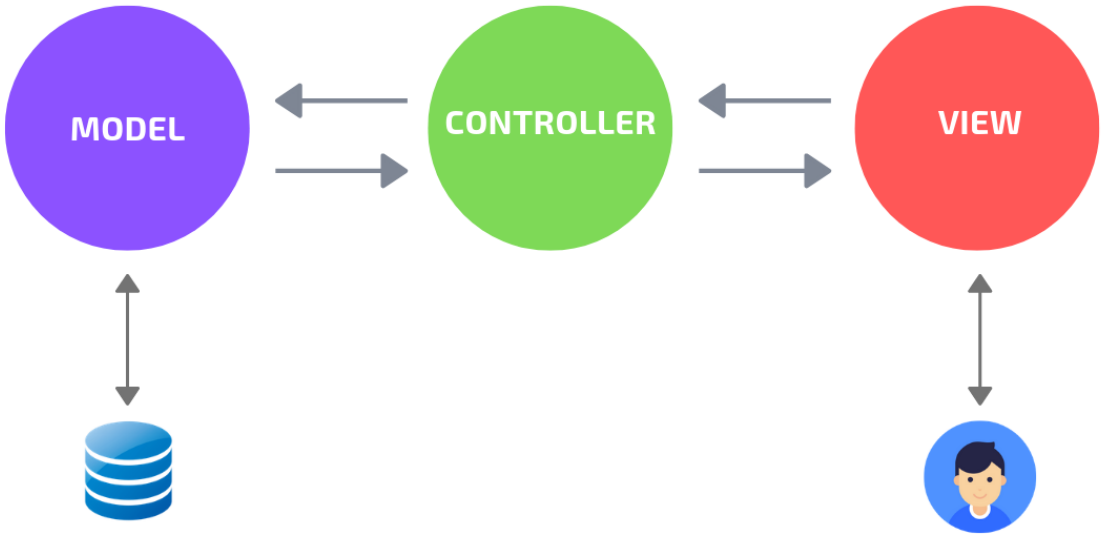


Figure 7. Estrutura MVC Laravel

12. Camadas do Laravel

O Laravel oferece uma estrutura de pastas pré-definida que facilita a organização do código e promove boas práticas de desenvolvimento, são elas:

- App: Núcleo da aplicação, aqui encontramos as camadas Models e Controllers.
  - Controllers: São responsáveis por processar requisições e interagir com Models e Views.
  - Models: Representam entidades e interagem com o banco de dados.
- Config: Armazena arquivos de configuração para diversos aspectos da aplicação, como banco de dados, autenticação e roteamento.

- Database: Registra a criação de Factories, Migrations e Seeders.
    - Factories: geram dados de teste de forma rápida e simples. Elas eliminam a necessidade de criar manualmente objetos de modelo com dados fictícios para cada teste unitário ou funcional.
    - Migrations: são essenciais para gerenciar o esquema do seu banco de dados de forma organizada e versionada. Para criar, modificar e remover tabelas, colunas e outros elementos do banco de dados.
    - Seeders: são classes PHP utilizadas para popular seu banco de dados de teste com dados iniciais.
- Public: Esta pasta é acessível diretamente pelo navegador web.
- Routes: Define as rotas da aplicação, mapeando URLs para Controllers.
- Storage: Armazena arquivos gerados pela aplicação, como uploads de arquivos.
- Tests: O local ideal para escrever testes unitários e funcionais para garantir a qualidade do código.

13. ORM (Object-Relational Mapper)

O Mapeador Relacional de Objetos é uma camada de abstração que facilita a interação entre a sua aplicação PHP e o banco de dados relacional, para definir Modelos e criar Classes PHP que representam tabelas do seu banco de dados.

- Realizar Consultas: utilizando métodos como find, where, select e outros.
- Inserir, Atualizar e Deletar Dados: Utilize métodos como create, save, update e delete para manipular dados nas tabelas.
- Definir relacionamentos de um-para-um, um-para-muitos e muitos-para-muitos entre os seus modelos, facilitando a consulta e manipulação de dados relacionados.

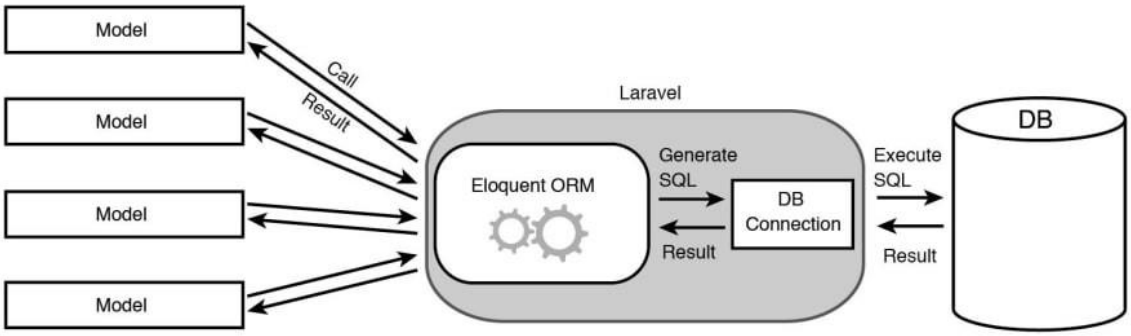


Figure 8. Representação Eloquent ORM

14. Repository Pattern

É um serviço que visa abstrair a lógica de acesso ao banco de dados, promovendo código reutilizável, manutenível e testável. Eles agem como uma camada intermediária entre a sua aplicação e o ORM Eloquent, encapsulando as operações CRUD (Create, Read, Update, Delete) e outras operações de negócio relacionadas ao acesso a dados de uma entidade específica (modelo).

14.1. Benefícios dos Repositories:

- Reuso de código: Evita a duplicação de código SQL e operações de acesso ao banco de dados em diferentes partes da aplicação. Ao centralizar essa lógica no Repository, você mantém o código limpo e organizado.
- Manutenção simplificada: Modificações na forma de interagir com o banco de dados para uma entidade ficam isoladas no Repository, facilitando atualizações futuras sem afetar outras partes da aplicação.
- Testabilidade aprimorada: Permite testar a lógica de acesso ao banco de dados de forma unitária, separadamente dos Controllers e Views. Isso garante a confiabilidade do código que manipula os dados.

• Separação de concerns: Mantém uma separação clara entre a camada de negócios (que utiliza os Repositories) e a camada de persistência (implementada com o Eloquent). Isso melhora a organização do código e sua legibilidade.

15. RTC - (Comunicação em Tempo Real) com Laravel

Esta tecnologia permite que usuários recebam atualizações instantaneamente sem a necessidade de recarregar a página. Essa comunicação é possibilitada por duas tecnologias: WebSockets e o pacote Laravel Echo.

15.1. WebSockets: a base da comunicação bidirecional

WebSockets são um protocolo de comunicação que permite uma conexão bidirecional persistente entre um navegador web e um servidor web. Diferente do HTTP tradicional, que é baseado em solicitações e respostas, os WebSockets mantêm a conexão aberta,

possibilitando a troca de dados em tempo real em ambas as direções (cliente-servidor e servidor-cliente).

O Laravel Echo é um pacote oficial que facilita a integração de WebSockets na sua aplicação. Ele fornece uma biblioteca JavaScript para o lado do cliente e um canal de eventos para o lado do servidor, permitindo que você:

- Inscrever usuários em canais: Os usuários se inscrevem em canais temáticos (por exemplo, um canal para atualizações de chat, outro para notificações).
- Emitir eventos do servidor: O servidor emite eventos para canais específicos, notificando usuários inscritos naquele canal.
- Processar eventos do cliente: O servidor pode processar eventos recebidos do cliente (por exemplo, enviar uma mensagem no chat).

16. Laravel Queue

É um recurso que permite executar tarefas demoradas assincronamente em segundo plano, adicionando tarefas em uma fila e o Laravel as processará posteriormente, sem afetar a resposta da sua aplicação ao usuário.

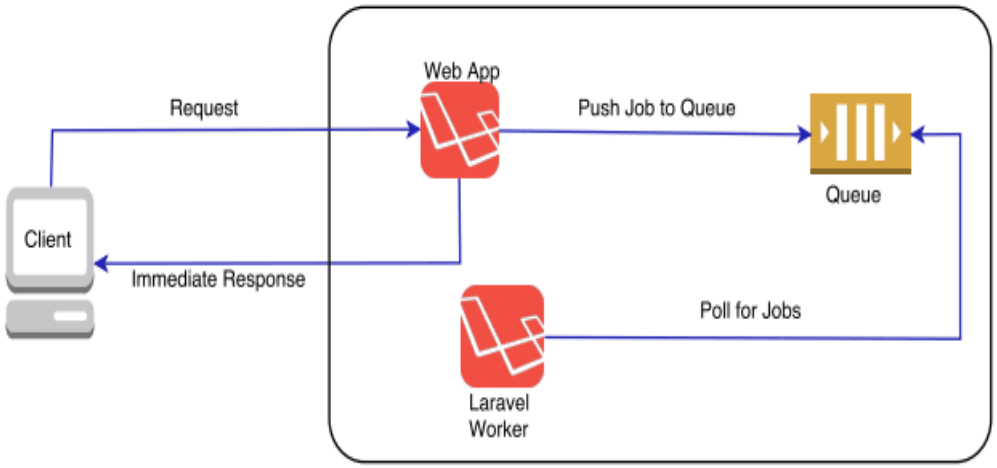


Figure 9. Representação Laravel Queue

16.1. Aplicações comuns com o Laravel Queue

- Envio de e-mails: Os envios de e-mails são adicionados em uma fila e processados em segundo plano, sem afetar o tempo de resposta do servidor.
- Processamento de imagens: As imagens são processadas em segundo plano, como no caso de aplicativos que precisam fazer o upload e a manipulação de muitas imagens em diferentes tamanhos e formatos.
- Atualizações de banco de dados: Importação de dados de outras fontes ou sincronização de dados com serviços externos, podem ser enfileiradas e processadas em segundo plano para evitar que bloqueiem o servidor web.
- Trabalhos em lote: Qualquer tarefa demorada que possa afetar o desempenho da aplicação pode ser beneficiada por ser enfileirada e processada assincronamente com o Laravel Queue.

17. Aplicação Swagger

A arquitetura do Swagger é composta por diversos elementos que trabalham em conjunto para fornecer uma solução completa para o design, documentação e geração de código de APIs.

17.1. Os principais componentes da arquitetura são:

- OpenAPI (Swagger 2.0): Padronizada a descrição de APIs RESTFUL, definindo endpoints, métodos HTTP, parâmetros, respostas e modelos de dados.
- AsyncAPI: Descreve as APIs assíncronas como baseadas em WebSockets ou eventos.
- JSON Schema: Define a estrutura de dados JSON para especificar os modelos de dados utilizados na API.

17.2. Ferramentas

Swagger Editor: Editor online para criar e editar manualmente definições de API em formato YAML ou JSON.

Swagger Codegen: Ferramenta para gerar código cliente e servidor a partir de definições de API Swagger.

Swagger Hub: Plataforma online para gerenciamento de APIs, incluindo documentação, colaboração, testes e publicação.

17.3. Processo de trabalho

Design de API: A API é projetada utilizando as especificações OpenAPI, AsyncAPI ou JSON Schema, definindo os endpoints, métodos HTTP, parâmetros, respostas e modelos de dados.

Documentação da API: A definição da API é utilizada para gerar automaticamente documentação detalhada e interativa, incluindo exemplos de uso e testes.

Geração de código: A definição da API pode ser utilizada para gerar código cliente e servidor em diversas linguagens de programação, facilitando o desenvolvimento de aplicações que consomem ou expõem a API.

17.4. Aplicações

Documentação de APIs RESTful: O Swagger é a ferramenta mais popular para documentar APIs RESTful, sendo utilizado por empresas de todos os portes.

Design colaborativo de APIs: O Swagger Hub facilita o design colaborativo de APIs, permitindo que diversos desenvolvedores trabalhem na mesma definição de API.

Geração de SDKs para APIs: O Swagger Codegen pode gerar SDKs para diversas linguagens de programação, facilitando o consumo da API por desenvolvedores.

Publicação de APIs em portais de desenvolvedores: O Swagger Hub permite publicar APIs em portais de desenvolvedores, facilitando a descoberta e o uso da API por outros desenvolvedores.

17.5. Exemplos de Aplicação

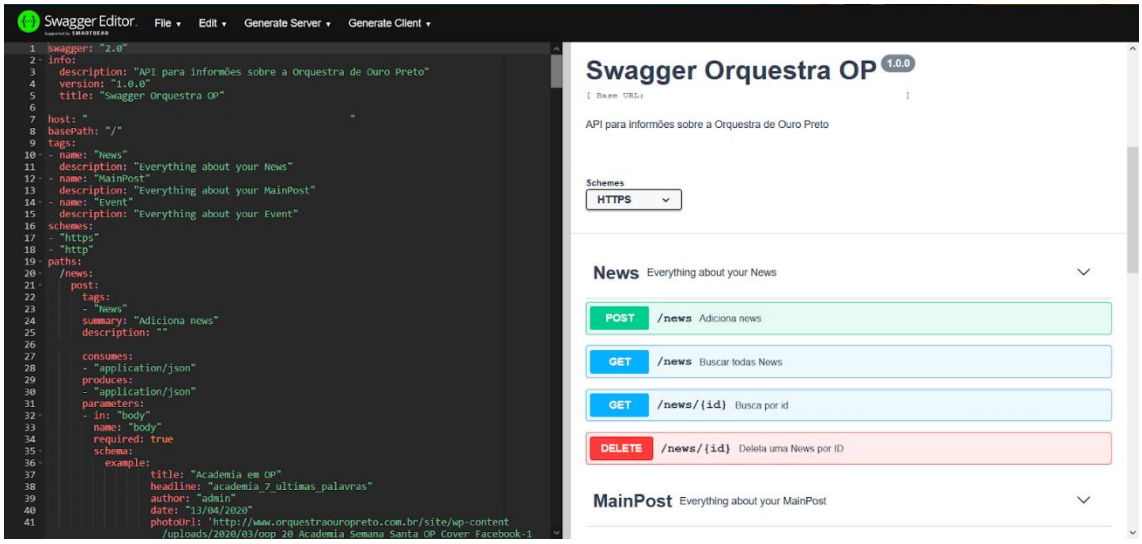


Figure 10. Swagger Orquestra OP - API

A Figura 10 mostra que o Swagger Editor está sendo usado para definir uma API para a Orquestra de Ouro Preto. A API fornece informações sobre a orquestra, como notícias, eventos e publicações principais. A API é definida usando a especificação OpenAPI 2.0, que é um formato padrão para descrever APIs RESTful.

Editor Swagger: No lado esquerdo da tela, há um arquivo de configuração YAML que define a estrutura da API, incluindo caminhos, operações e parâmetros.

Interface Swagger UI: No lado direito da tela, se trata da representação visual interativa da API. É possível explorar e testar os endpoints da API de maneira interativa.

Seções exibidas: Podemos visualizar parcialmente as seções da API, sendo listada a seção “News”. Esta seção tem métodos GET e POST para os endpoints “/news” e “/news/{id}”.

```php
<?php

namespace App\Documentation;

use Exception;

class SwaggerGenerator
{
    private array $documentationClass = [];

    public function registerDocumentationClass(SwaggerBase $documentation): void
    {
        $this->documentationClass[] = $documentation;
    }

    public function generationOpenApiArray(): array
    {
        if (empty($this->documentationClass)) {
            throw new Exception('Nenhuma classe de documentação foi registrada');
        }

        $baseClassInstance = $this->documentationClass[0];

        $openApiDocument = $baseClassInstance->toArray();

        $openApiDocument['paths'] = [];

        foreach ($this->documentationClass as $documentationClass) {
            $paths = $documentationClass->getPaths();
            $openApiDocument['paths'] = array_merge($openApiDocument['paths'], $paths);
        }

        return $openApiDocument;
    }

    public function generateOpenApiJson(): string
    {
        $openApiArray = $this->generationOpenApiArray();
        $jsonContent = json_encode($openApiArray, JSON_PRETTY_PRINT);
        $filePath = storage_path('app/public/api-docs.json');

        file_put_contents($filePath, $jsonContent);
        return $jsonContent;
    }
}
```

Figure 11. Código SwaggerGenerator.php

Análise do Figure 11

A classe SwaggerGenerator define três métodos públicos:

- registerDocumentationClass(): Este método registra uma classe de documentação com a instância SwaggerGenerator, responsável por definir os recursos da API.
- generationOpenApiArray(): Este método gera um array que representa o documento OpenAPI.
- generateOpenApiJson(): Este método gera um JSON que representa o documento OpenAPI.

Geração do JSON OpenAPI

O método generateOpenApiJson() chama o método generationOpenApiArray() para gerar um array que representa o documento OpenAPI.

Em seguida, o método converte o array em JSON usando a função json_encode().

Por fim, o método grava o JSON em um arquivo chamado api-docs.json.

OBSERVAÇÕES

- O código usa comentários para explicar o propósito de cada método e variável. Isso torna o código mais fácil de entender e manter.
- Usa a função file_put_contents() para gravar o JSON em um arquivo. Esta função é útil para salvar o documento OpenAPI para que ele possa ser usado por outras ferramentas.

RECURSOS ADICIONAIS:

- Site oficial do Swagger: https://swagger.io/
- Documentação da especificação OpenAPI: https://www.openapis.org/
- Documentação da especificação AsyncAPI: https://www.asyncapi.com/
- Documentação da especificação JSON Schema: https://json-schema.org/

18. Versionamento do Banco de Dados

Aplicando as boas práticas no ambiente de desenvolvimento STIC, por padrão todas as soluções desenvolvidas devem obrigatoriamente seguir como padrão o versionamento do banco de dados em aplicações PHP. Em resumo, fica estabelecido em caráter obrigatório o uso do comando migrate para gerar as migrações, evidenciando a evolução/histórico do banco de dados, conforme podemos analisar na figura abaixo:

Name	Last commit	Last update
..		
2022_03_25_153350_create_consumidores_table.p...	Criação CRUD tb_consumidor	2 years ago
2022_03_25_153905_create_consumidoreslps_tabl...	crud tb_consumidor	2 years ago
2022_03_25_154007_create_servicos_olinda_table....	crud tb_consumidor	2 years ago
2022_03_25_154208_create_logs_table.php	crud tb_consumidor	2 years ago
2022_03_25_171832_create_tb_consumidor_servic...	crud tb_consumidor	2 years ago
2022_04_25_153350_alter_consumidores_table.php	Ajuste consumidor	2 years ago
2022_04_25_171251_alter_consumidores_token_ta...	Ajustando validação de acessos	2 years ago
2022_04_26_183113_alter_servicos_olinda_table.php	Ajustando validação de acessos	2 years ago
2022_05_02_141213_alter_tb_consumidores_table....	Ajustes	2 years ago
2024_01_15_000000_alter_servicos_pronta_olinda_...	Correção do erro data	3 months ago
2024_02_16_000000_alter_log_olinda_table.php	Adicionado log para a api	3 months ago

Figure 12. Demonstração de Versionamento do Banco de Dados

Na Figure 12, temos condições de analisar o histórico de migrações realizadas em um banco de dados específico e em qualquer momento podemos realizar um Rollback em estados anteriores, para reverter situações pontuais. Esse recurso é fundamental para versionar a incrementação de novas tabelas ou inserções de novos atribuições e correções.

De forma clara é exibido uma lista de commits de um repositório. Cada entrada inclui a data, o número do commit, a descrição do commit, quem fez o último commit e quando ele foi atualizado pela última vez. As descrições indicam várias etapas de desenvolvimento, como a criação de serviços, tabelas, ajuste de validação de acesso e correção de erros de dados. Isso é relevante para os desenvolvedores ou equipes que estão rastreando mudanças em seus projetos de software ao longo do tempo.

2022_03_25_153350_create_consumidores_table.php 1.14 KB

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateConsumidoresTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('tb_consumidor', function (Blueprint $table) {
            $table->increments('id_consumidor');
            $table->date('data_inicio');
            $table->date('data_fim');
            $table->string('client_id');
            $table->string('client_user');
            $table->string('client_password');
            $table->string('num_sei');
            $table->string('email_responsavel');
            $table->string('nome_responsavel');
            $table->string('telefone_responsavel');
            $table->timestamps();

            /**
             * This part is necessary for us to use the soft-delete feature.
             */
```

Figure 13. Migration consumidores

Na Figure 13, temos como exemplo um arquivo PHP com um código para gerar a migração de banco de dados em um ambiente Laravel. O código faz parte da classe Illuminate\Support\Facades\Schema e inclui uma função chamada “up()” para criar uma nova tabela com as colunas: ‘id’, ‘data_inicio’, ‘data_fim’, ‘client_id’, ‘client_user’, ‘client_password’, ‘num_sei’, ‘email_responsavel’, ‘nome_responsavel’ e ‘telefone_responsavel’.