Rodrigo Felippe Alves Cabral

Integração com API REST e TOTVS: Um estudo de caso de um PDV

Vilhena - RO

2022

Rodrigo Felippe Alves Cabral

Integração com API REST e TOTVS: Um estudo de caso de um PDV

Trabalho de conclusão de curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – *Campus* Vilhena, como requisito à obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Vilhena - RO
2022

FICHA CATALOGRÁFICA

Biblioteca IFRO – Campus Vilhena

C116i

CABRAL, Felippe Rodrigo Alves
Integração com API REST e TOTVS : um estudo de caso de um PDV / Felippe Rodrigo Alves Cabral – Vilhena, Rondônia, 2022.

85f. ; il.
Orientador Prof. M.e. Roberto Simplício Guimarães

Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO

1. TOTVS 2. IFRO 3ADVPL 4. ERP 5. PDV I. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – IFRO II. Título

006.7882

Bibliotecária responsável Rosilene Maria do Couto Marques CRB 11/321

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA
Vilhena - Código INEP: 1056974
Rodovia BR 174, KM 3, nº4334, CEP 76982-270, Vilhena (RO)
CNPJ: 10.817.343/0003-69 - Telefone: 69 2101-0703

ATA DE DEFESA DE MONOGRAFIA

Na data 22/11/2022 realizou-se a sessão pública de defesa da Monografia intitulada **API INTEGRAÇÃO PDV: Uma API REST de integração com software TOTVS** apresentada pelo aluno **Rodrigo Felippe Alves Cabral (2017105053032-6)** do Curso **Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (Vilhena)**. Os trabalhos foram iniciados às **14:00** pelo Professor **Roberto Simplicio Guimaraes** presidente da banca examinadora, constituída pelos seguintes membros:

- **Roberto Simplicio Guimaraes** (Orientador)
- **Marco Antonio Augusto de Andrade** (Coorientador e Examinador Interno)
- **Gilberto Pereira da Silva** (Examinador Interno)

A banca examinadora, tendo terminado a apresentação do conteúdo da Monografia, passou à arguição do candidato. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo aluno, tendo sido atribuído o seguinte resultado:

[X] APROVADO **Nota: 91**

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu **Roberto Simplicio Guimaraes** lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

VILHENA / RO, 22/11/2022

Documento assinado eletronicamente por **Rodrigo Felippe Alves Cabral**, Discente, em 24/11/2022, às 14:59, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Roberto Simplicio Guimaraes**, Orientador, em 24/11/2022, às 14:44, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marco Antonio Augusto de Andrade**, Examinador Interno, em 24/11/2022, às 15:03, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Gilberto Pereira da Silva**, Examinador Interno, em 24/11/2022, às 14:42, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Dedico este trabalho e minha formatura a minha mãe, Maria Auxiliadora Alves, que me deu a vida e que sempre lutou para que eu estudasse e tivesse uma graduação, sei que esse dia demorou mas chegou...

Agradecimentos

Os meus agradecimentos vão primeiramente a Deus, o Criador, agradeço a toda minha família em especial as minhas filhas Gabriele Alves e Elisa Alves e minha esposa Anelize Appelt, a pessoa que mais me incentivou a fazer este curso. Agradeço a todos os professores do curso de ADS do Instituto Federal de Rondônia e outros que se envolveram comigo nesse caminho, que compartilharam muito de seus conhecimentos e me deram força para concluir. Agradeço também a todos os colegas de turma que tive o prazer de conhecer nessa jornada, tanto os que concluíram o curso como os que ficaram pelo meio do caminho, e agradeço aos meus colegas de trabalho, em especial a meu amigo Lee Chardes que me ensinou os primeiros passos na área de desenvolvimento de software.

Resumo

A tecnologia da informação, cada vez mais evidenciada nos últimos anos, tem trazido diversos benefícios na vida cotidiana e vem sendo uma exigência no mundo dos negócios. O Software Protheus é um sistema de ERP desenvolvido pela TOTVS e, de acordo com seu site (TOTVS, 2022), é utilizado no Brasil e em diversos países da América Latina, apresenta algumas limitações em alguns processos, que hoje só são disponíveis através de uma estação de trabalho fixa. Este trabalho tem como objetivo viabilizar a interação com outras plataformas através do desenvolvimento de uma API de integração responsável por abrir inúmeras possibilidades, deixando disponíveis, por exemplo, os processos de cadastro de clientes e produtos, emissão de orçamento, pré-vendas ou comandas eletrônicas, entre outras, conforme necessidade do cliente, independentemente da forma ou linguagem utilizada para tal. Portanto esse trabalho consiste no desenvolvimento de uma API que será construída utilizando uma Arquitetura Orientada a Serviços, com ênfase na arquitetura REST fazendo uso de *Web Services*, permitindo que aplicações que venham a ser implementadas em diferentes plataformas e linguagens de programação possam consumir os serviços. A implementação da API faz uso da linguagem TL++ e da plataforma de serviços do software de ERP TOTVS Protheus, armazenando e consumindo dados do banco de dados do software, e irá dispor de documentação com os *endpoints* implementados. A implementação dessa API garantiu uma inovação para o software ERP em questão e possibilita a integração entre diferentes empresas e soluções de tecnologia, mediando serviços e somadas à capacidade de troca e volume de dados entre várias plataformas.

Palavras-chave: IFRO, ADVPL, TOTVS, Protheus, API, REST.

Abstract

Information technology, increasingly evident in recent years, has brought several benefits in our lives and has been a requirement in the business world. The Protheus Software, a TOTVS Protheus ERP system developed by TOTVS and according to its website (TOTVS, 2022), is used in Brazil and in several Latin American countries. from a fixed workstation. This work aims to enable interaction with other platforms through the development of an integration API responsible for opening up numerous possibilities, making available, for example, customer and product registration processes, budget issuance, pre-sales or electronic commands. , among others, according to the client's needs, regardless of the form or language used for this purpose. However, this work proposes the development of an API that will be built using a Service Oriented Architecture, with emphasis on the REST architecture making use of Web Services, allowing applications that will be implemented in different platforms and programming languages to consume the services. The API implementation makes use of the TL++ language and the services platform of the ERP software TOTVS Protheus, storing and consuming data from the software database, and will have documentation with the implemented endpoints.

Keywords: IFRO, ADVPL, TOTVS, Protheus, API, REST, Restful, mobilidade, integração.

Lista de ilustrações

Lista de Quadros

Lista de tabelas

Lista de abreviaturas e siglas

ADVPL	Advanced Protheus Language
ANSI	American National Standards Institute
ASCII	American Standard Code For Information Interchange
API	Application Programming Interface
BPM	Business Process Model
CTO	Chief Technology Officer
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MVP	Minimum Viable Product
OMT	Object-Modeling Technique
OOSE	Object-Oriented Software Engineering
PDV	Ponto de Venda
REST	Representational State Transfer
RPO	Repositório de Objetos
SOAP	Simple Object Access Protocol
SGDB	Sistema de Gestão de Bases de Dados
TCP	Transmission Control Protocol
TL++	TOTVS Language Plus Plus
UML	Unified Modeling Language
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language

Sumário

1 Introdução

1.1 Contexto e problema

Neste mundo tecnológico, diz-se atualmente que a economia é informacional, visto que a produtividade e competitividade das empresas estão diretamente ligadas às suas capacidades de gerar, processar e aplicar de forma consciente a informação baseada em conhecimento como diz Castells (1999), softwares estão cada vez mais inseparáveis da economia e da vida cotidiana. Observa-se no mundo dos negócios, que a tecnologia já se tornou aliada fundamental tanto para as atividades finalísticas quanto para a gestão, de tal forma que hoje é inviável alguma empresa se manter competitiva no mercado sem a utilização de nenhum tipo de tecnologia.

> *As empresas administram muitos ativos, pessoas, dinheiro, instalações, mas a informação talvez seja o seu maior ativo e o que lhe causem maior perplexidade. Negócios requerem mudanças constantes ao passo que os sistemas, uma vez implementados, permanecem relativamente rígidos.* (WEILL P.; ROSS, 2006)

Os sistemas de informação segundo Castells (1999), vem para auxiliar na organização das informações, fazendo com que os dados sejam utilizados com maior eficiência, facilitando a tomada de decisão, melhorando a produtividade do trabalho e consequentemente, aumentando a eficácia da gestão. Surge, então, como opção para as organizações, um novo produto denominado ERP - Enterprise Resource Planning ou Planejamento de Recursos Empresariais, são concebidos para atuar em todos os momentos do negócio. Os sistemas integrados (ERP) compõem um fenômeno presente há duas décadas no panorama global empresarial, podendo ser aplicados com pequenas adaptações a qualquer empresa (SHIH Y. E HUANG, 2009). São capazes de integrar dados de todos os setores de uma organização, se tornando assim, cada vez mais indispensáveis para as empresas que desejam evoluir e otimizar suas operações.

Para Christensen (2012), as inovações de softwares podem ser classificadas como incrementais ou disruptivas. O autor deixa claro que as inovações incrementais são mais comuns e representam melhorias no desempenho de produtos já existentes, normalmente representam uma mudança contínua de adequação a uma nova tecnologia ou demanda dos clientes.

Neste cenário, as APIs (do inglês, Interface de Programação de Aplicação), permitem a comunicação entre componentes ou softwares através de padrões e protocolos. E como

cita (SAUDATE, 2013) as API são um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços.

Este trabalho se propõe a desenvolver uma API de integração para o sistema de ERP TOVS Protheus, visando possibilitar a comunicação entre plataformas para agilizar algumas funcionalidades, como cadastros de clientes e produtos, emissão de orçamento, pré-vendas ou comandas eletrônicas, entre outras, conforme necessidade do cliente, permitindo o desenvolvimento de facilitadores orientados a processo, através de qualquer plataforma ou dispositivo, de modo que os colaboradores possam cadastrar e consultar clientes, produtos e preços, adicionar itens vendidos e instantaneamente enviar todas essas informações para o caixa, o que assegura maior celeridade ao processo do início ao fim do atendimento, e com isso contribui para uma melhor experiência do cliente.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver uma API de integração com Software Protheus da TOTVS

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalhos são:

- Analisar o processo de pré-vendas e orçamentos juntos aos usuários chaves, avaliando os requisitos da API;
- Desenvolver uma API utilizando o modelo de arquitetura REST, com a finalidade disponibilizar uma integração com qualquer plataforma;
- Escrever a documentação da API com o intuito de facilitar o entendimento dos desenvolvedores, possibilitando que outras aplicações possam consumir a mesma;
- Realizar testes de validação da API com ferramentas e plataformas de consumo web.

1.3 Justificativa

Inovações incrementais são mais comuns e representam melhorias no desempenho de produtos já existentes, normalmente representam uma mudança contínua de adequação a uma nova tecnologia ou demanda dos clientes é o que deixa claro Christensen (2012). Segundo informações de Saudate (201) a implantação da integração via API nas empresas

tem crescido muito nos últimos anos, devido às facilidades que proporcionam às organizações, a utilização de *Web Services* tradicionais como o SOAP (*Simple Object Access Protocol*) não apresenta um bom desempenho em aplicações móveis, pois estes tipos de *Web Services* são baseados em trocas de XML, para este cenário isto seria muito custoso, pois existe um consumo de elevado de recursos de infra-estrutura de serviços, hardware e internet. Sistemas ERP em geral, segundo Junior (2012), são modularizados e tem seus processos pré-definido, e a utilização de uma API abriria a possibilidade de flexibilizar processos e agilizar atividades.

2 Fundamentação teórica

Nesta seção serão apresentados o software utilizado como base para a criação dessa API, como também uma breve explanação sobre sistemas ERP, além das principais tecnologias, padrões, ferramentas utilizadas e definições para implementação do modelo arquitetural REST.

2.1 UML

A UML, Linguagem de Modelagem Unificada (Unified Modeling Language) segundo Guedes (GUEDES, 2011) surgiu da união de três métodos de modelagem: o método Booch, o método OMT (Object Modeling Technique) e o método OOSE (*Object-Oriented Software Engineering*) e é uma linguagem visual utilizada para modelar softwares baseados no paradigma de orientação a objetos. Por ser uma linguagem de propósito geral no projeto utilizamos o UML para representar casos de uso e diagramas de classe. E assim Gosala define os diagramas classes como:

> *...um tipo de diagrama UML que ajuda a expressar as classes e relações entre as classes. Diagramas de classes (CD) UML são usados para projetar e ilustrar a estrutura do software. São ferramentas muito importantes para engenheiros entenderem a estrutura básica de um sistema. (GOSALA, 2021)* (TANEBAUM, 2010)

Gosala (2021) exauta o uso de diagramas UML, *"diagramas UML são muito vantajosos para pesquisadores, desenvolvedores de software e acadêmicos"*, utilizados para *"estudar, analisar, documentar, projetar e desenvolver qualquer software eficientemente..."*.

2.2 HEFLO

A ferramenta HEFLO é "Solução BPM (Business Process Model) para Modelagem e Automatização de Processos de Negócio" (HEFLO, 2022) como detalha a sua desenvolvedora Venki em seu site. A Solução foi utilizada para a criação gráfica do fluxo de processos de ciclos de vida da aplicação proposta neste trabalho.

2.3 TAIGA

De acordo com seu site (TAIGA, 2022) o Taiga é um Open-source Project Management Software, com sua tradução livre sendo definido como um Software de Gerenciamento de Projetos de Código Aberto. É, na verdade uma plataforma que possui várias ferramentas para auxiliar equipes multidisciplinares no desenvolvimento de softwares de maneira ágil. O Taiga nasce em 2013, depois que a equipe da Kaleidos (sua desenvolvedora) não encontrou uma ferramenta que atendesse a necessidade de sua equipe de desenvolvedores e designers.

- **Método Kanban**

  Kanban é um método para definir, gerenciar e melhorar serviços que empregam trabalho do conhecimento, ele pode ser utilizado mesmo que outro método esteja sendo utilizado. Segundo Camargo e Ribas o KanBan foi concebido por David Anderson em meados de 2007 inspirado na filosofia lean de gestão, originária da Toyota (CAMARGO ROBSON E RIBAS, 2019). No método kanban o fluxo de trabalho é mapeado em uma sequência de passos que representam as etapas pelas quais os itens passam.

2.4 Gerenciamento de configuração e mudanças com GITLAB

Observou-se em pesquisas realizadas que a grande maioria dos projetos de desenvolvimento de software é comum e natural que mudanças sejam aceitas e ocorram de acordo com as necessidades apresentadas no decorrer dos processos deixa entendido Camargo e Ribas (2019). Partindo desse conhecimento surgiu o Gerência de Configuração de Software ou apenas GSM, que é um conjunto de atividades de apoio que permitem controlar as mudanças que ocorrem no desenvolvimento de software, mantendo a estabilidade na evolução do projeto (O'GRAND, 2018). Em resumo, o gerenciamento de configuração e mudança nos diz as mudanças, os motivos e o resultado de tais mudanças realizadas no artefato. Para esse gerenciamento utilizamos a Ferramenta GitLab (GITLAB, 2022).

O GItlab é, pelo que é possível entender nas palavras de de OGrand (2018) uma plataforma web de hospedagem de código e rastreamento de problemas baseada no sistema de controle de versão GIT. Adam O'Grand (2018) registra em seu livro o lançamento do Gitlab pela primeira vez em 2011, continuou a crescer e evoluir ao longo dos anos, adicionando novos recursos e capacidades, e se transformou em uma ferramenta de uma etapa para uma força de trabalho ágil. Embora seja de propriedade e gerenciado pela GitLab Inc., que orienta a direção do projeto, o núcleo do GitLab é um software de código aberto com mais de 2.000 contribuidores separados até o momento. (O'GRAND, 2018)

2.5 Sistemas ERP

De acordo com Cícero Júnior (JUNIOR, 2012), os sistemas de informações nas organizações empresariais modernas tornaram-se condição de sobrevivência a partir da década de 1990, o que no princípio era apenas uma vantagem competitiva empresarial hoje é considerado mais uma ferramenta aplicada ao dia a dia dos responsáveis pelas tomadas de decisões. O termo “Sistemas ERP” é um termo que de forma didática conceitua um sistema de informação adquirido na forma de pacotes comerciais de software que permite a integração de dados entre todos os processos de negócios de uma organização, assim são definidos os sistemas de gestão (JUNIOR, 2012). Em tradução livre da sigla ERP (*Enterprise Resource Planning*) obtemos em português o nome "Planejamento dos Recursos Empresariais", o que não define perfeitamente a funcionalidade desses sistemas, pois eles vão além da atuação somente no planejamento, já que eles controlam e fornecem suporte a todos os processos operacionais, produtivos, administrativos e comerciais de uma organização.

Em um artigo publicado pelo site brasileiro Portal ERP (PORTALERP, 2022), sistemas ERP são definidos como m software corporativos, que tem a principal finalidade de oferecer um suporte para que as empresas possam obter controle total de suas informações. Em geral, os software ERP são divididos em camadas, e JUNIOR (JUNIOR, 2012) separa essas camadas em 3:

Aplicação: Nesta camada temos o software ERP com as suas funcionalidades, processos, cadastros (formulários divididos em campos) e demais dados necessários para a operação da empresa. Os dados gerados na camada "Aplicação"devem ser armazenados de forma lógica no Banco de Dados (Possivelmente algum software ERP tenha que acessar o Banco de Dados por um mecanismo de conexão, não sendo uma conexão nativa);

Configuração e painel de controle: Todo software ERP deve ter uma camada onde é possível configurar/parametrizar o sistema e também customizar/personalizar o ERP, para isso é necessária uma camada de construção de novo código-fonte e sua compilação, para que, assim, estas novas funcionalidades desenvolvidas fora do ERP padrão estejam disponíveis na aplicação;

***Framework* desenvolvimento:** Onde estão os códigos-fontes, tabela de banco de dados e onde é possível desenvolver novas rotinas, aplicar atualizações e gerenciar o código personalizável, tudo que foi desenvolvido fora do padrão.

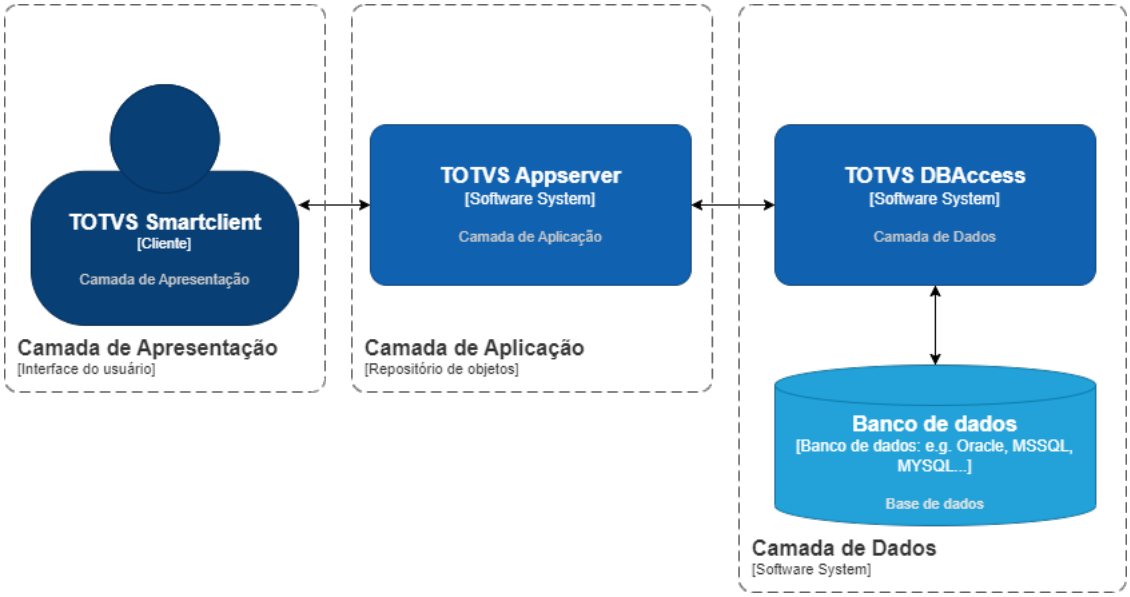
Já Davenport (DAVENPORT, 1998) apresenta as funcionalidades dos sistemas ERP separando-as em funções internas chamadas *BackOffice*, compostos por recursos humanos, manufatura e finanças, e funções externas nomeadas *Front-office*, que são compostas por vendas e serviços, além da tecnologia e do gerenciamento da cadeia de suprimentos.

2.6 TOTVS Protheus

De acordo com o seu site (TOTVS, 2022), em Abril de 1991 a Microsiga Software início o desenvolvimento do produto Siga Advanced. Nesta época o sistema era desenvolvido sobre a linguagem Clipper 5.x. O sistema ERP Protheus foi oficialmente lançado em 1998, uma remodelagem do antigo sistema. O software foi idealizado na intenção de ser uma alternativa nacional aos ERP´s de empresas estrangeiras, nos anos seguintes, a Microsiga investiu na evolução do seu sistema e, líder de mercado, comprou outras empresas como Datasul, Logocenter e RM, formando a TOTVS, cita Marcello Nuzzi, gestor de serviços do segmento de manufatura da TOTVS e especialista no Protheus. A TOTVS hoje é uma multinacional brasileira que atende mais de 30 mil clientes no país que criou sua linguagem de programação própria, o ADVPL (Advanced Protheus Language) e que posteriormente veio se tornar o TL++.

Assim, o sistema TOTVS Protheus é um ERP que tem a sua estrutura dividida em camadas:

Figura 1 – Infraestrutura do Software

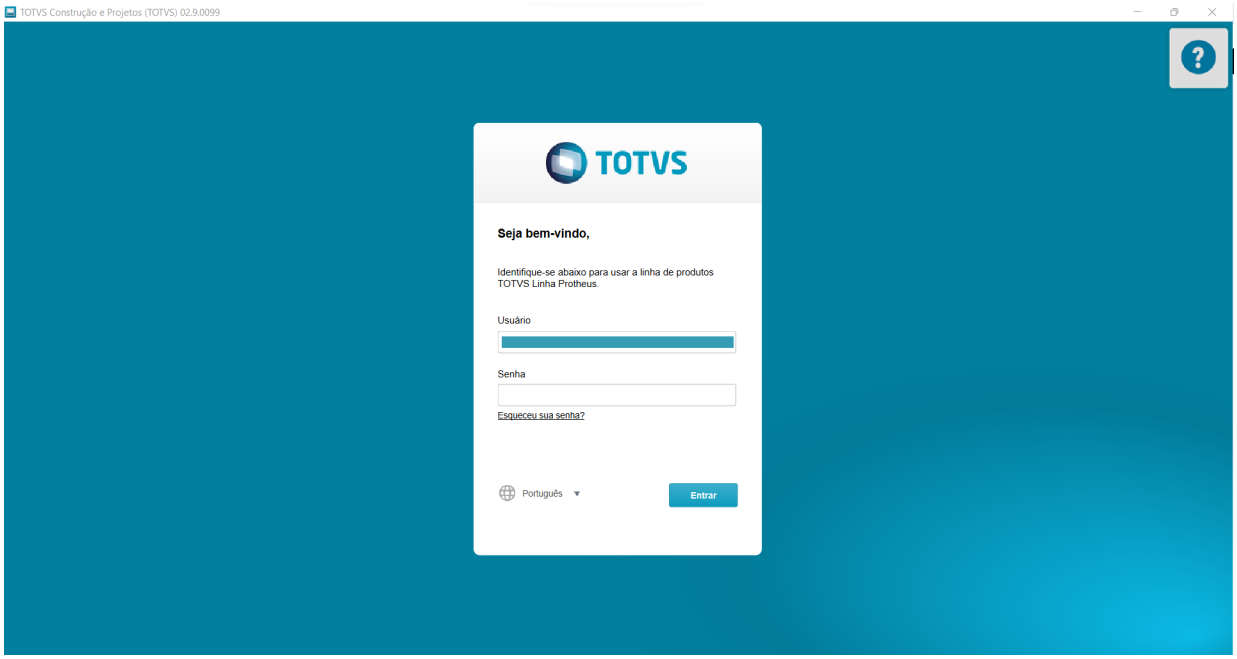


Fonte: Produzido pelo autor

- **Camada de Apresentação:**

  O TOTVS | SmartClient é um software desenvolvido em C++ e é responsável pela camada de apresentação multiplataforma, a interface do usuário. O SmartClient conta com versões x86(32 bits) e x64 (64Bits) de acordo com informações disponíveis no site de seu desenvolvedor (TOTVS, 2022). Podemos ver a imagem na figura 2.

Figura 2 – Captura de tela aplicação TOTVS Protheus SmartCLient

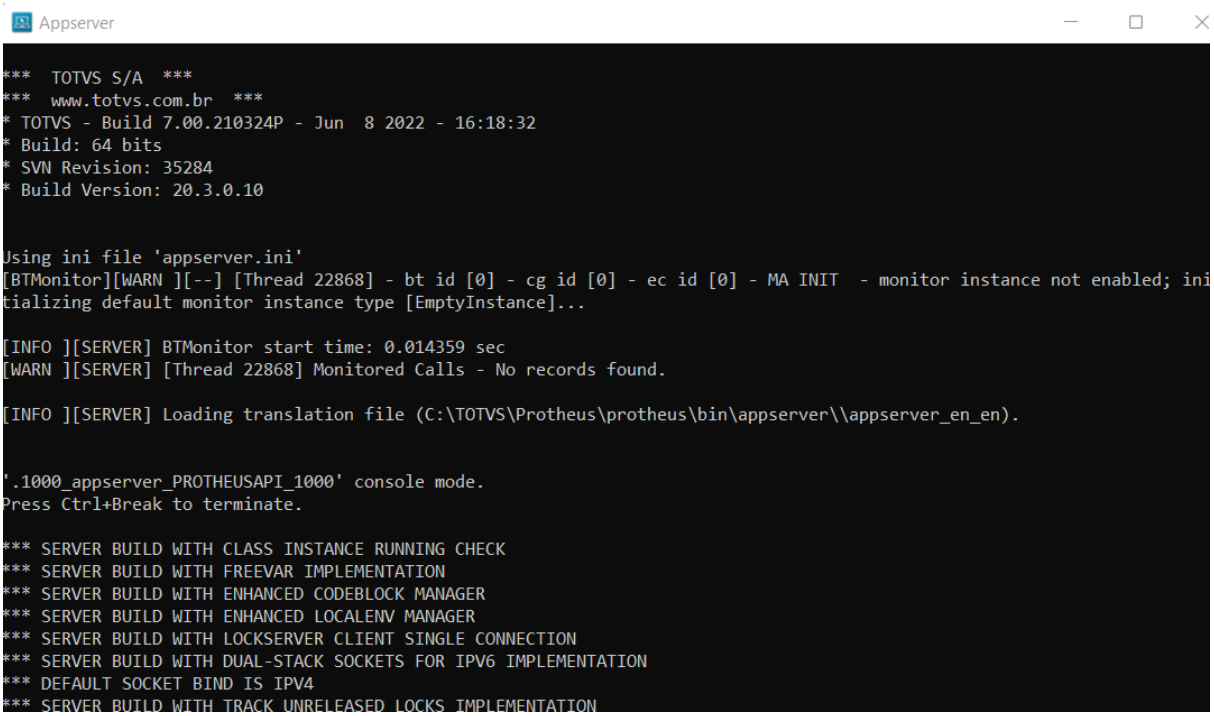


Fonte: captura realizada pelo autor

- **Camada de Aplicação:**

  O TOTVS | Application Server foi desenvolvido em ANSI C++ e, portanto, o núcleo do Appserver pode ser recompilado em todos os sistemas operacionais e plataformas que suportem ANSI C++, segundo seu site (TOTVS, 2022).

Figura 3 – Captura de tela aplicação TOTVS Protheus Appserver



Fonte: captura realizada pelo autor

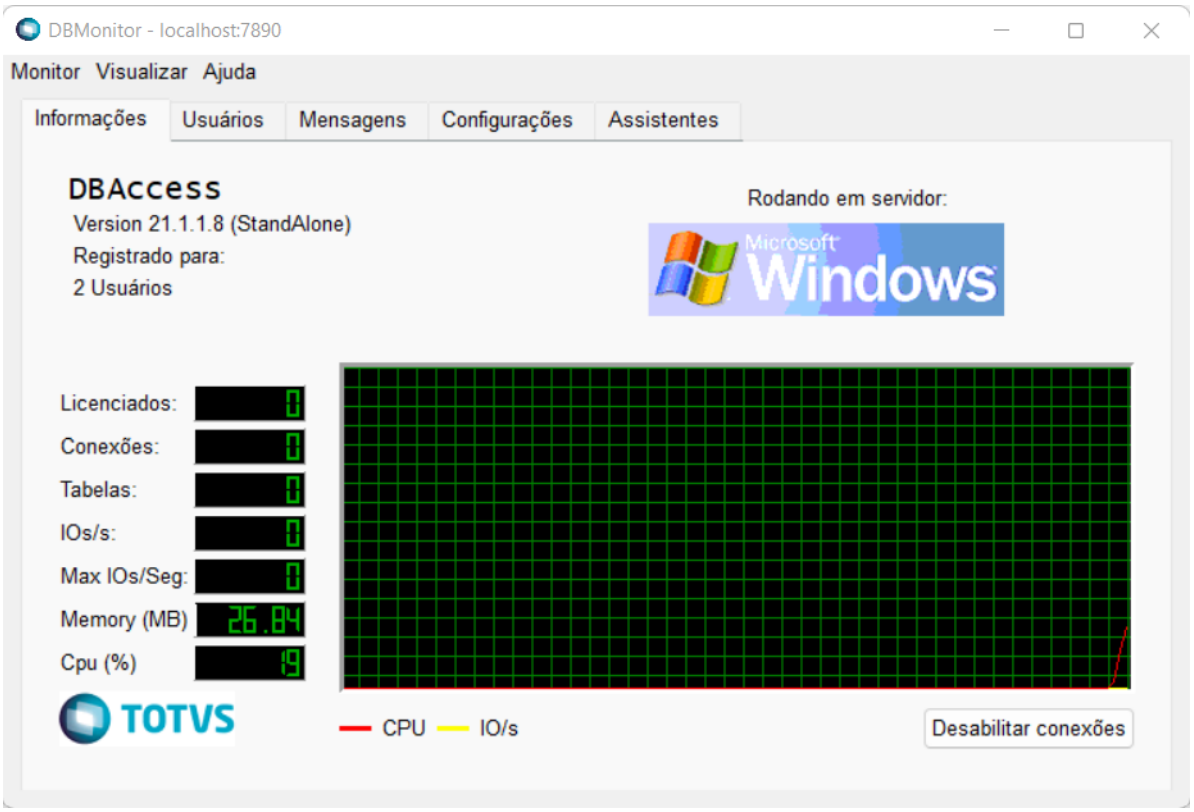
O Repositório de objetos Protheus é a estrutura do sistema onde são armazenados suas diversas funções, comandos, estruturas e operações que são, de certa forma, a base fundamental de todo sistema. Os APO's como são chamados o RPO – Repositório de Objetos Protheus, é onde é possível incluir atualizações ou personalizações realizadas no sistema. É também o local onde estão todas as funções, patchs e informações, carregadas de forma dinâmica pelo arquivo de configuração do TOTVS | Application Server o AppServer.ini para que as rotinas sejam executadas pelo TOTVS Protheus de acordo com as definições de ambiente e instancias.

- **Camada de dados:**

  O TOTVS | DBAccess é uma ferramenta de conexão com banco de dados que segundo o desenvolvedor permite o acesso a uma grande variedade de SGBDs sem a necessidade de geração de códigos específicos para cada um deles.

  O TOTVS | DBMonitor é uma ferramenta gráfica que permite configurar o ambiente que será utilizado para o acesso ao banco de dados além de permitir a visualização e monitoramento das conexões realizadas entre o TOTVS Appserver e o TOTVS DBAccess.

Figura 4 – Captura de tela aplicação TOTVS DBMonitor



Fonte: captura realizada pelo autor

2.7 Linguagem ADVPL e TL++

O ADVPL é uma linguagem de programação desenvolvida pela Microsiga Software em 1994, para criação de sistemas integrados de gestão ERP. Essa linguagem é baseada no padrão xBase um dos primeiros SGBDs do mercado, considerado pai de linguagem como Clipper, dbFast, xHArbor entre outros. O ADVPL contem comandos, operadores, funções, controles de fluxos e palavras reservadas que permite o desenvolvimento a de maneira procedural ou através da implementação do conceito d orientação a objetos. (TOTVS, 2022).

O TOTVS Protheus possui suas funções reservadas (Function), onde apenas desenvolvedores da TOTVS conseguem compilar, porém está disponível a todos os usuários a possibilidade de desenvolvimento de Funções de Usuários (User Funcions) Classes e Webservices.

```
INCLUDE 'TOTVS.CH'

User Function HelloWorld()
    Local cMsg := "Hello World!"
    Conout(cMsg)
Return
```

Quadro 2.1 – Exemplo código ADVPL

A função exibida acima, é um exemplo de função desenvolvida em ADVPL, essa função escreve na tela do console da aplicação o texto "Hello World!"

Para a integração através de webservices com foi desenvolvida uma API de integração através da Linguagem TL++. A linguagem TL++ (TOTVS Language Plus Plus, ou também chamada de TLPP) é uma evolução do AdvPL, desenvolvida pela TOTVS para trazer ao programador AdvPL algumas facilidades vistas em outras linguagens. (TOTVS, 2022).

2.8 JSON (JavaScript Object Notation)

De acordo com Crockford (2022), o protocolo Notação de Objetos JavaScript (JSON) é uma formatação leve, de troca de dados. Para seres humanos é fácil de ler e escrever. Para máquinas é fácil de interpretar e gerar. Está baseado em um subconjunto da linguagem de programação JavaScript, Standard ECMA-262 1a Edição - Dezembro 1999. JSON é em formato texto e completamente independente de linguagem, pois usa convenções que são familiares as linguagens C e familiares, incluindo C++, C, Java, JavaScript, Perl, Python e muitas outras. Estas propriedades fazem com que JSON seja um formato ideal de troca de dados. O Protocolo JSON possui uma estrutura de fácil compreensão e foi o modelo de dados utilizado para integração das informações entre as plataformas. Abaixo vemos um exemplo de código em JSON onde visualizamos como um arquivo no formato JSON é estruturado:

```
{
  "codigo": "001",
  "nome": "Rodrigo Felippe",
  "limite": 3000.1,
  "dataDeCadastro": "2014-06-25T00:00:00.000Z",
  "ativo": true,
  "dados": {
    "rg": 12345,
    "cpf": "123456789890",
    "telefones": {
      "casa": "33221100",
```

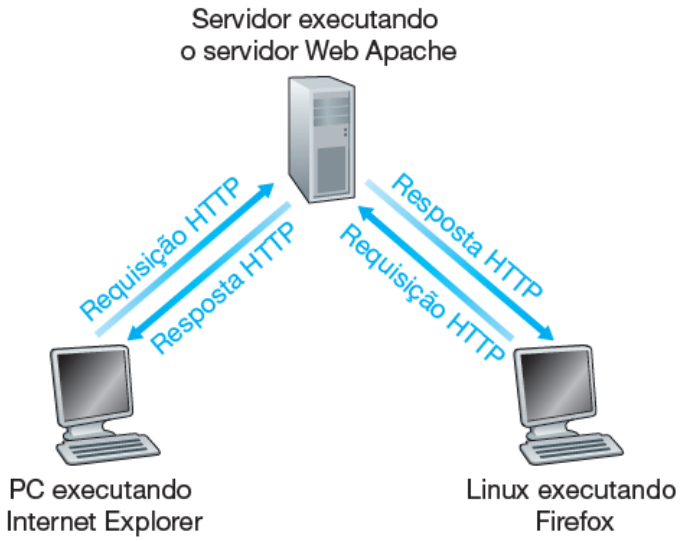
```
        "celular": "99887766"
      }
    }
}
```

Quadro 2.2 – Exemplo de arquivo JSON

2.9 Hypertext Transfer Protocol (HTTP)

O HTTP *Hypertext Transfer Protocol* ou Protocolo de Transferência de Hipertexto em português, é um protocolo que especifica como é a comunicação entre um navegador e um servidor web, sendo um dos principais da *World Wide Web* (WWW), surgiu nos primeiros anos da década de 1990. (ROSS, 2013) explicam que o protocolo da camada de aplicação, é executado na forma de cliente servidor. Os dois executados em sistemas finais diferentes, porém, conversam entre si por meio de troca de mensagens HTTP.

Figura 5 – Diagrama requisição HTTP



Fonte: (ROSS, 2013)

O HTTP determina como os usuários (Clientes) requisitam páginas web aos servidores e como eles as transferem aos clientes. O HTTP usa como o protocolo de transporte TCP. Embora o HTTP tenha sido projetado para utilização na Web, Tanenbaum (TANEBAUM, 2010) cita que ele foi criado de modo mais geral que o necessário, tendo em vista às aplicações futuras orientadas a objetos. Com esse motivo, são aceitas operações chamadas métodos, diferentes da simples solicitação de uma página da Web. Cada

requisição é feita em formato de texto ASCII, sendo a primeira linha o nome do método solicitado, por exemplo: GET, POST ou PUT, diferenciando-se letras minúsculas e maiúsculas:

```
GET nomearq HTTP/1.1
```

Quadro 2.3 – Exemplo requisição http.

De acordo também com Tanenbaum (2010) os métodos internos do HTTP são os seguintes:

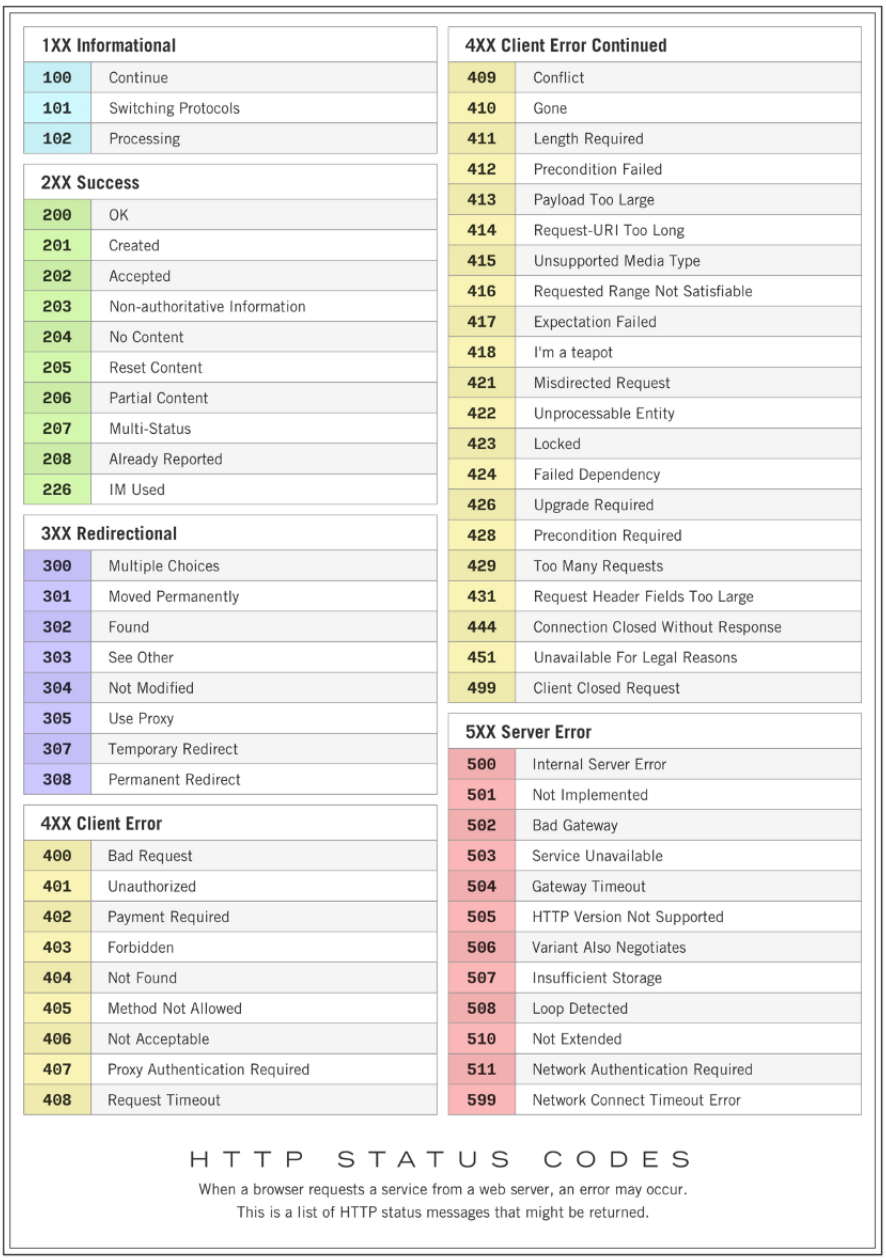
Tabela 1 – Métodos HTTP.

Método	**Descrição**
GET	Solicita a leitura de uma página da Web
HEAD	Solicita a leitura de um cabeçalho de página da Web
PUT	Solicita o armazenamento de uma página da Web
POST	Acrescenta a um recurso (por exemplo, uma página da Web)
DELETE	Remove a página da Web
TRACE	Ecoa a solicitação recebida
CONNECT	Reservado para uso futuro
OPTIONS	Consulta certas opções

> *As aplicações Web REST baseiam-se no protocolo HTTP e utilizam seus principais métodos POST, GET, PUT e DELETE para operar sobre os recursos. Cada método está relacionado a uma função, com alguma diferença no método PUT que pode ser usado por mais de uma maneira.* (TANEBAUM, 2010)

Toda requisição HTTP recebe uma resposta que consiste em uma linha de status e possivelmente informações adicionais, como por exemplo uma página web. A linha de status é formada por um código de 3 dígitos que informa ao cliente se a requisição foi atendida, e caso não tenha sido, os motivos delas. Na figura 6 é possível visualizar as famílias de status que podem ser recebidas como resposta.

Figura 6 – Códigos de resposta HTTP



1XX Informational

100	Continue
101	Switching Protocols
102	Processing

2XX Success

200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional

300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error

400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

4XX Client Error Continued

409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error

500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

Fonte: Retirado de https://www.steveschoger.com/

2.10 API

API - Application Programming Interface (Interface de Programação de Aplicação) são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos.

As APIs são uma maneira simplificada de conectar a própria infraestrutura por

meio do desenvolvimento de aplicações nativas em nuvem. No entanto, elas também possibilitam o compartilhamento de dados com clientes e outros usuários externos. As APIs públicas agregam valor de negócios porque simplificam e ampliam a forma como você se conecta aos parceiros, além de, possivelmente, monetizar seus dados. Um exemplo famoso é a API do Google Maps, como relata (REDHAT, 2022).

2.11 *Representational State Transfer* (REST) e RESTFul

O REST é definido por Saudate (SAUDATE, 2013) como um modelo arquitetural e de boas práticas, que foi desenvolvido por um dos criadores do HTTP, o doutor RoyFielding, no início dos anos 2000.

O objetivo geral é a padronização é definir o conjunto de melhores práticas, chamadas de *constraints*, com a ideia de determinar a forma que os padrões como *Hypertext Transfer Protocol* (HTTP) e *Uniform Resource Identifier* (URI), também chamados de *EndPoint*, devem ser modelados, a fim de aproveitar dos recursos oferecidos.

De acordo com (MARQUES, 2018) conclui-se que a REST é uma ótima alternativa para casos com limitações de recursos e de largura de banda. Além disso, é flexível, pois não apresenta restrições no formato em que a informação retorna, mas no comportamento dos componentes. Na figura **??** é possível ver um gráfico de utilização desse modelo de desenvolvimento criado por (MUSSER, 2011) em relação aos outros modelos existentes.

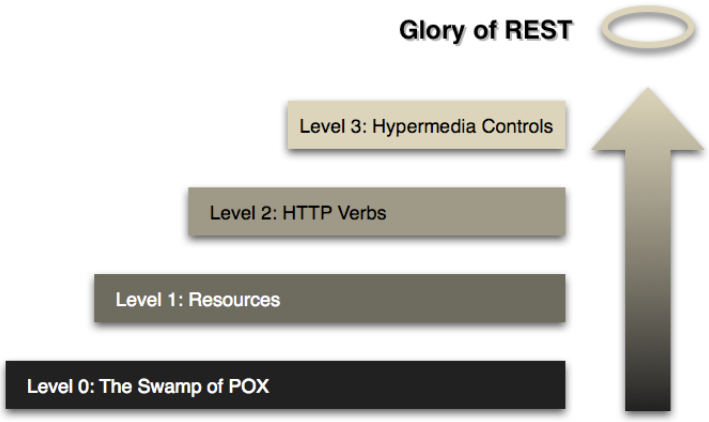
Existem seis restrições que foram pensadas para constituir o estilo arquitetural. As constraints do REST definidas por Marques (MARQUES, 2018) são:

- ***Client-Server:*** É a restrição básica para uma aplicação REST. O objetivo desta divisão é separar a arquitetura e responsabilidades em dois ambientes, possibilitando o isolamento de forma organizada para cada uma das funcionalidades.
- ***Stateless:*** Possibilidade cliente mandar várias requisições para o servidor, porém, cada uma delas devem ser independentes.
- ***Cacheable:*** É necessário que estas respostas possam ser cacheadas, evitando processamento desnecessário.
- ***Uniform Interface:*** São pequenas regras para deixar um componente o mais genérico possível, o tornando muito mais fácil de ser refatorado e melhorado, dentro dessa regra existe uma forma para fazer essa comunicação uniforme, como por exemplo, cada *resource* deve ter um endpoint específico e coeso, uma representação do *resource*, resposta autoexplicativa e retornar todas as informações necessárias na resposta para que cliente saiba navegar.

- ***Layered System:*** A sua aplicação deve ser composta por camadas, e estas camadas, o cliente nunca deve chamar diretamente o servidor da aplicação sem antes passar por um intermediador.
- ***Code-On-Demand:*** Permitir que o cliente possa executar algum código sob demanda, ou seja, estender parte da lógica do servidor para o cliente.

Já o RESTFul é visto por Saudate (SAUDATE, 2013) como algo concreto diferentemente da abstração que é o modelo REST, é a implementação do conceito e boas práticas do REST em uma determinada API, assim uma API é considerada RESTFul. Para que uma API seja considerada RESTFul, temos o modelo de maturidade de Leonard Richardson, onde 4 níveis são apresentados para que seja definido a maturidade da aplicação, detalhados na figura 7.

Figura 7 – Níveis de maturidade REST



Fonte: Retirado de https://martinfowler.com/articles/richardsonMaturityModel.html

- ***Level* 0:** Quando a API utiliza o protocolo HTTP apenas como comunicação.
- ***Level* 1:** Utiliza os endpoints de forma eficiente, apresentando um mapeamento definido de seus recursos.
- ***Level* 2:** Utiliza a semântica correta de seus verbos HTTP (GET, POST, PUT e DELETE) e os códigos de retorno padrões como definidos na figura 6.
- ***Level* 3:** Após contemplar todos os níveis anteriores e principalmente utilizar HATEOS, cujo objetivo é ajudar os clientes a consumirem o serviço sem a necessidade de conhecimento prévio profundo da API, apresentando a relações de links pra recursos.

2.12 Postman

O Postman (POSTMAN, 2022), criado em 2012, é muito utilizado pelos desenvolvedores de *WebServices* e APIs para realizar pedidos através de métodos HTTP, esse software oferece uma interface simplificada e intuitiva, que facilita os testes e a depuração dos serviços REST, sendo possível realizar solicitações HTTP como *GET, POST, PUT e DELETE*. Além disso, pode-se realizar o teste de performance de uma API executados um conjunto de solicitações para verificar seu desempenho de resposta. Toda documentação da API é gerada após a conclusão e parametrização dos testes.

3 Materiais e métodos

Esta seção aborda a questões relacionadas à integração de informações através de API de Integração com o Protheus, o estudo realizado para elaboração de uma taxonomia com o intuito de categorizar as necessidades de integração das informações do sistema ERP e a API proposta. São expostos os serviços que a API oferece e como ela está estruturada, destacando os seus principais requisitos através de um diagrama de classe. Além disso, a modelagem adotada, o desenvolvimento de uma aplicação, por fim, a forma como os dados são armazenados no sistema.

3.1 Ciclo de vida e processos

A metodologia utilizada para desenvolvimento do software é o controle de fluxo de processos de produção, Kanban, com suas fases divididas em cartões de acordo com a sua etapa. Foram feitos levantamentos para cada uma das fases do ciclo de vida do software e esses cartões foram adicionados com uma cor representando cada etapa do processo. Três fases foram definidas, sendo elas: Inicialização, Execução e Finalização.

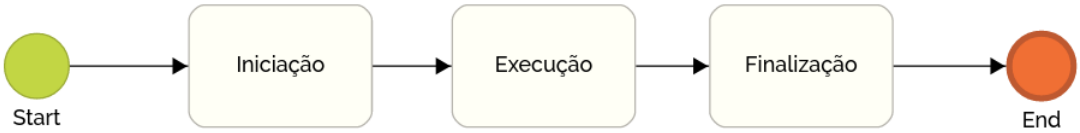
O Cronograma do ciclo de vida foi definido conforme a tabela abaixo:

Tabela 2 – Cronograma do Ciclo de vida.

Fase	Início	Fim
Iniciação	01/04/2022	31/05/2022
Execução	01/06/2022	31/08/2022
Finalização	01/09/2022	10/11/2022

A inicialização consiste na fase de definição de escopo e levantamento de requisitos, a execução é a parte de codificação de acordo com as definições da fase de inicialização e por último a fase de finalização que pode ser definida como a documentação do projeto de desenvolvimento. A figura 8 exemplifica de maneira macro o ciclo de vida de software:

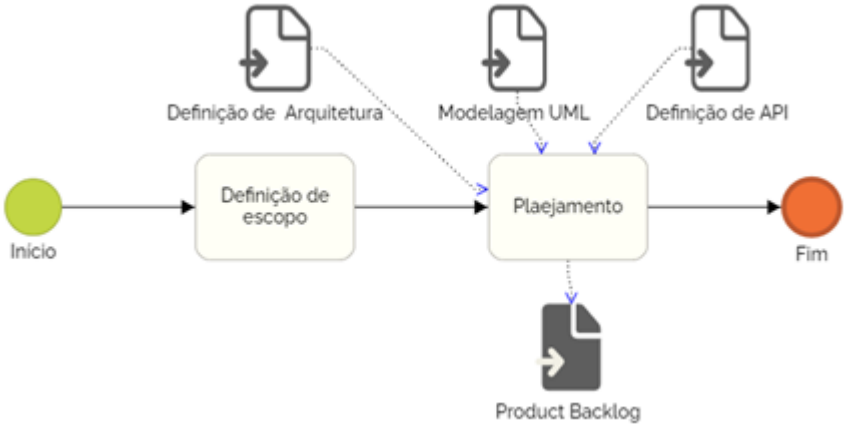
Figura 8 – Ciclo de vida



Fonte: elaborado pelo autor

A primeira etapa do ciclo é a iniciação, de acordo com a figura 9 do diagrama da etapa de iniciação, é definido o escopo do aplicativo que será desenvolvido, nessa etapa iremos realizar o planejamento onde tem como premissa definir a arquitetura, realização da modelagem UML e definição da API, resultando no produto *backlog*.

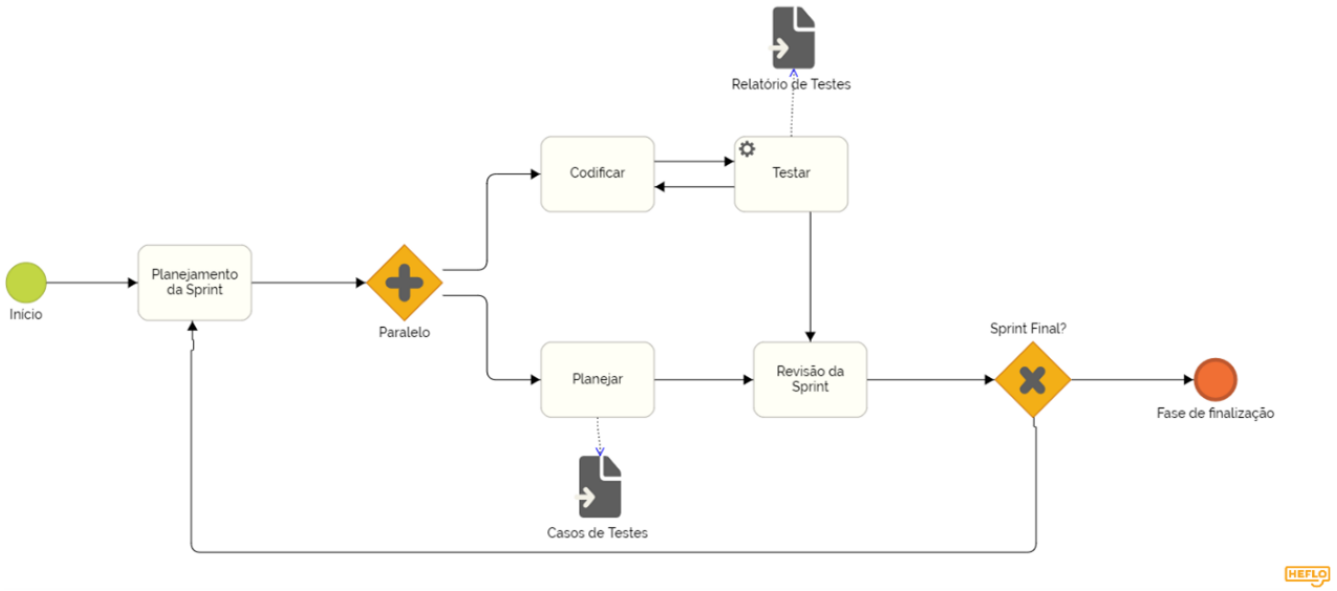
Figura 9 – Fase de iniciação



Fonte: elaborado pelo autor

A fase de execução inicia-se pela etapa de planejamento da execução de acordo com a atividades definida. O item a ser desenvolvido, após essa etapa duas etapas em paralelo se inicia, a codificação e os testes. A codificação resulta no código fonte do projeto, de então os testes são executados. Após esse passo, feito o *review* da atividades, e definindo-se sua finalização, passando a próxima atividade até a conclusão do MVP, Produto Mínimo Viável, do ingles *Minimum Viable Product*. A figura 10 exemplifica de maneira gráfica essa fase.

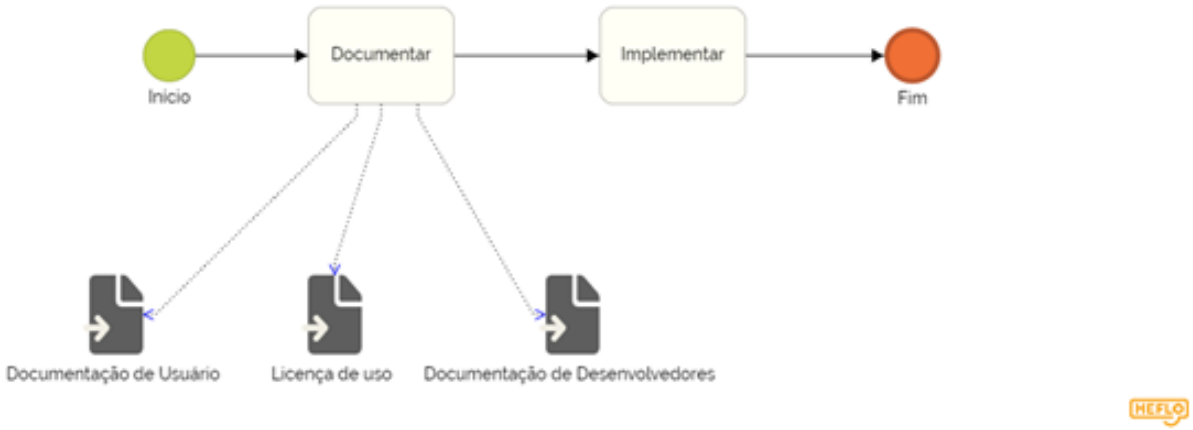
Figura 10 – Fase de execução



Fonte: elaborado pelo autor

A última fase do ciclo de vida é a finalização. Nessa etapa são realizados os processos de documentação e implementação da ferramenta, gerado-se a documentação de uso para o usuário, a licença de uso da API e a documentação dos desenvolvedores.

Figura 11 – Fase de finalização



Fonte: elaborado pelo autor

3.2 Requisitos

3.2.1 Requisitos Funcionais

- **RF01 – Manter produto**

  A API deve incluir, consultar e alterar o cadastro de produtos que fica localizado no sistema ERP.

- **RF02 – Manter cliente**

  O sistema deve incluir, consultar e alterar o cadastro dos clientes.

- **RF03 – Realizar orçamento**

  A API deve ter a opção de incluir uma orçamento através de POST na API.

3.2.2 Requisitos não-funcionais

Os requisitos não funcionais foram divididos em requisitos referentes à segurança, disponibilidade e tecnologia.

- **RNF01 – Disponibilidade**

  Todos os *endpoints* do sistema expostas como *webservices* poderão ser acessadas por sistemas externos. Este acesso precisa ser seguro.

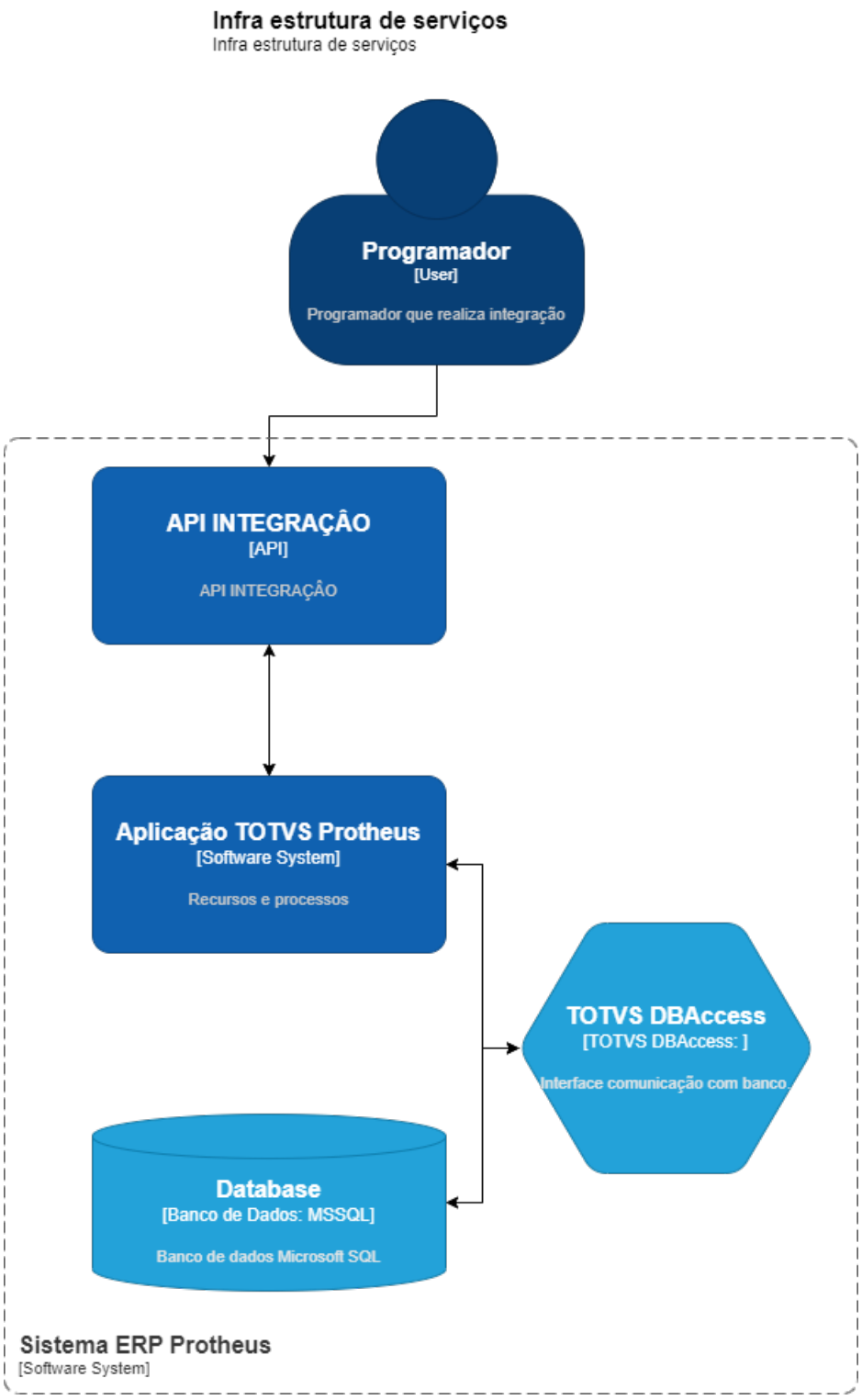
- **RNF02 –API deve ser considerada RESTful**

  A API deve implementação do conceito e boas práticas do REST.

3.3 Infraestrutura dos serviços

A infraestrutura dos serviços exibidas na figura 12, mostra a localização da API no modelo de aplicações definidas para maiorias dos usuários desse modelo de software.

Figura 12 – Infraestrutura dos serviços

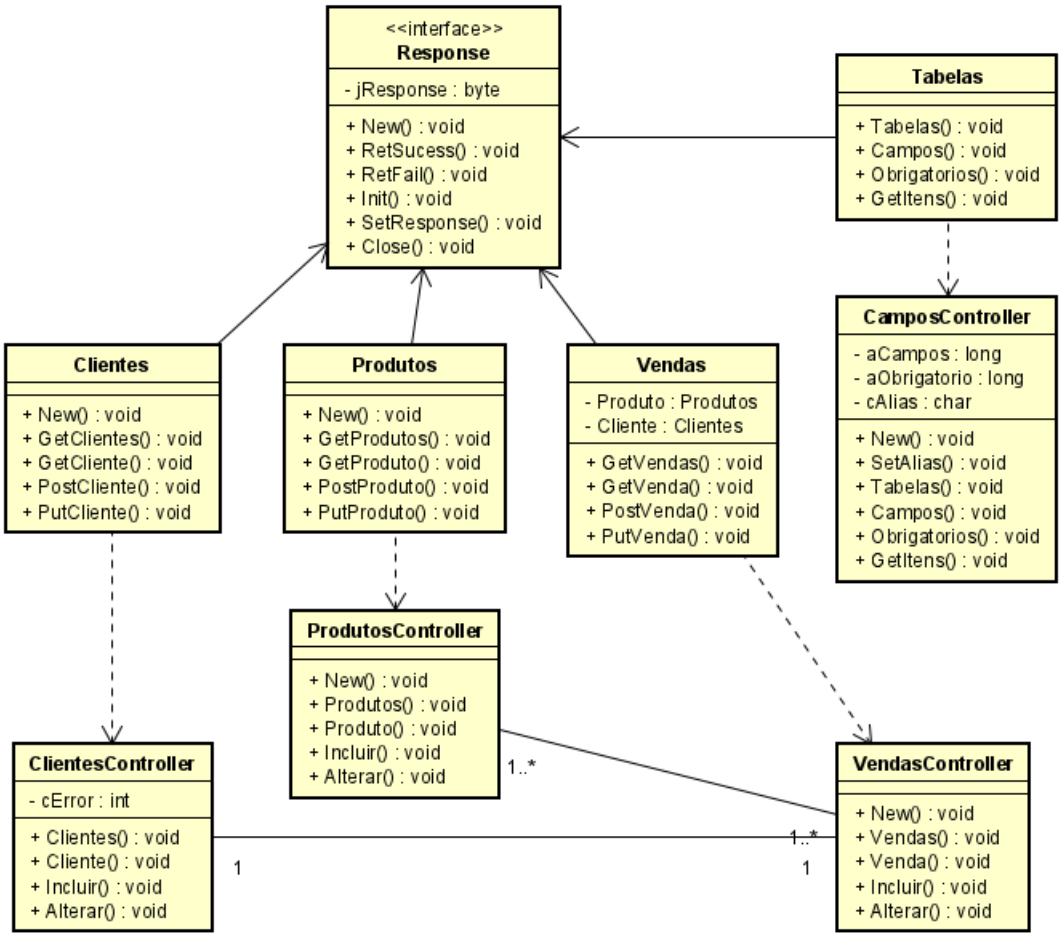


Fonte: elaborado pelo autor

3.4 Modelagem

A modelagem UML desse projeto, conta com o diagrama de classe na figura 13, esse diagrama foi desenvolvido no momento da iniciação do projeto, O diagrama de classes ilustra graficamente como será a estrutura do software e a relações das classes que servirão de modelo para objetos dessa API.

Figura 13 – Diagrama de classe



Fonte: elaborado pelo autor

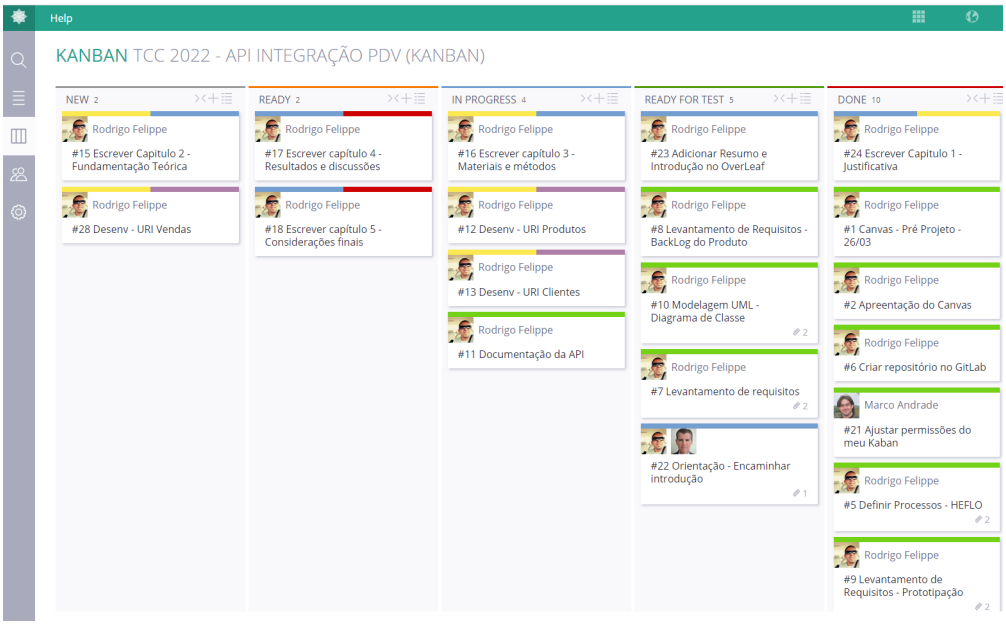
3.5 Metodologia Ágil

Na fase de iniciação foi definido que o processo de desenvolvimento seria gerenciado utilizando duas ferramentas do Taiga, a primeira ferramenta utilizada foi o KANBAN, para controle geral do projeto de maneira geral.

3.5.1 Kanban

Na utilização da metodologia Kanban, definimos 5 quadros de fluxo de atividades de acordo com seus status. Assim, através desta ferramenta, viabilizou-se a atribuição de prioridades para implementação das funcionalidades e, até mesmo, a inclusão conforme a necessidade iam surgindo no decorrer da elaboração. Dessa maneira, tornou-se possível o gerenciamento das tarefas, além da organização do tempo necessário para cumprir as metas, reduzindo os riscos/custos e aumentando a produtividade. Ademais, utilizou-se de um TAGS (Marcadores) de componentes para uma melhor representação de cada tarefa.

Figura 14 – Software Taiga - KanBan



Fonte: Captura de tela software Taiga

Na figura 14 é possível os seguintes quadros com as historias de usuários, ou atividades a serem realizadas e suas colunas que definem o fluxo de processos que foram separados nas seguintes etapas:

- ***New*** – Demandas definidas são adicionadas, para definição de *TAGS*.
- ***Ready*** – Pronta para partir para execução
- ***In Progress*** – Em execução
- ***Done*** – Demanda concluída

Já as *TAGS* foram divididas em dois conjuntos, o primeiro conjunto é o tipo de gênero das atividades, exibindo se é uma atividade de documentação ou atividade de desenvolvimento.

Tabela 3 – Gênero de atividade.

TAG(Marcador)	Descrição
	Documentação
	Desenvlvimento
	Testes

O segundo conjunto de *TAGS* são referentes a fase do ciclo de vida do projeto, de acordo com capítulo 3.1.

Tabela 4 – Cliclo de vida

TAGS(Marcador)	Descrição
	Iniciação
	Execução
	Finalização

3.6 Plano de testes

Os testes da API serão realizados através da plataforma POSTMAN, onde as requisições são feitas em todos os URI´s definidos neste trabalho. O resultado dos testes apresentados de acordo com capitulo 4.1.

3.7 Regras de Negócios

Para que seja possível a definição de um orçamento ou venda para a aplicação, deve-se definir as regras de negócio no contexto do sistema. Assim, pensando no cenário do TOTVS Protheus, em que para realização de uma operação é basicamente necessário a capacidade de cadastrar, consultar, atualizar e remover seus produtos de uma base de dados de da empresa, sendo assim, foram definidas as regras a seguir:

1. Um usuário pode consultar, cadastrar ou alterar um cliente.
2. Um usuário pode cadastrar ou alterar um produto.
3. Um usuário pode consultar, cadastrar ou alterar uma venda.

3.8 Definições dos *Definição de Endpoints*

Com as regras de negócio definidas, pode-se estabelecer os endpoints necessários para as requisições da API. Na Tabela 5 a seguir, são mostradas as rotas, métodos e finalidades de cada tipo requisição:

Tabela 5 – *Endpoints*

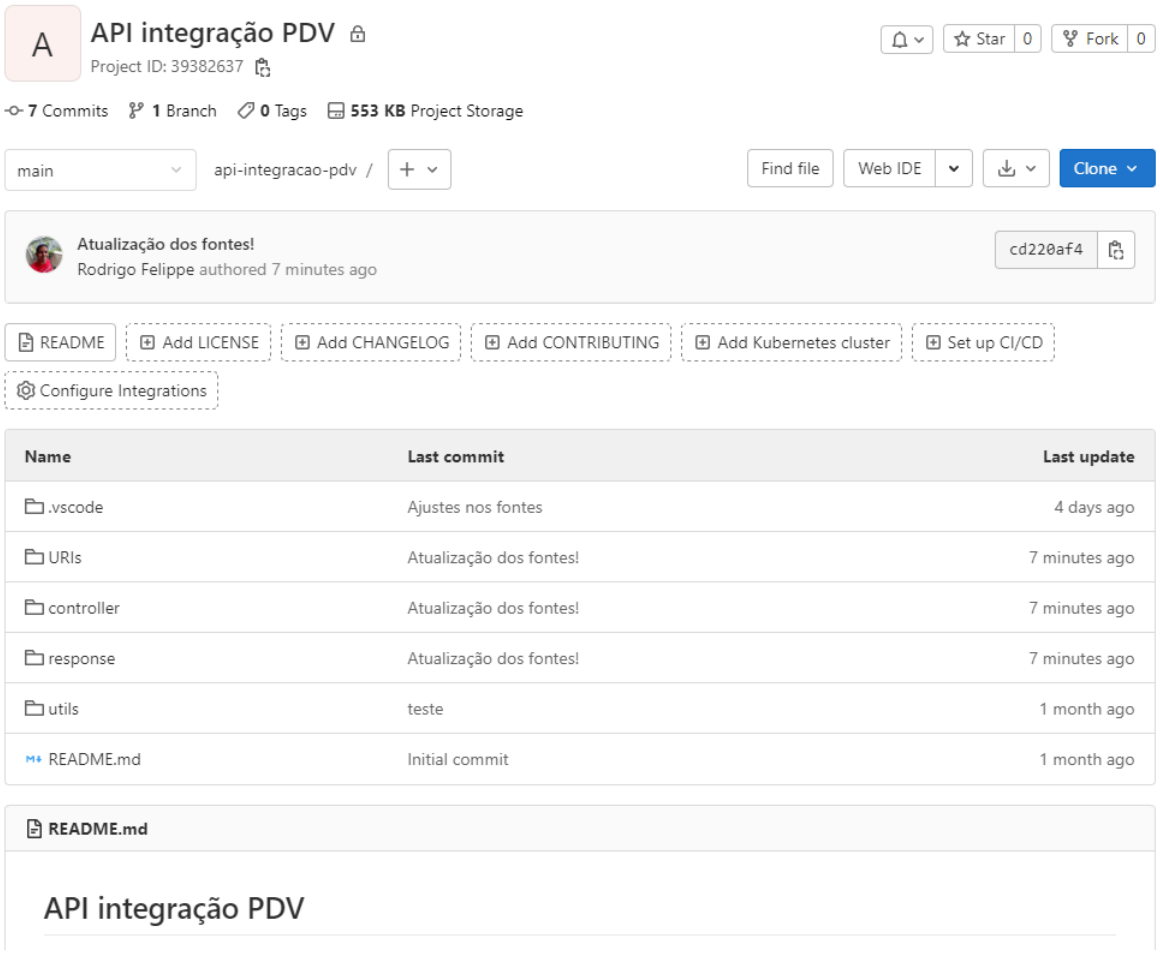
Endpoints	**Requisição**	**Descrição**
/produtos	GET	Consulta informações de produtos
/produtos/incluir	POST	Cadastro de Produtos
/produtos/:PRODUTO	GET	Consulta detalhes dos produtos
/clientes	GET	Consulta informações de clientes
/clientes/incluir	POST	Cadastro de clientes
/clientes/:CLIENTE	GET	Consulta detalhes dos produtos
/clientes/:CLIENTE/alterar	PUT	Altera cadastro de produto
/vendas	GET	Consulta informações de vendas
/vendas/incluir	POST	Cadastro de vendas
/vendas/:CLIENTE	GET	Consulta detalhes das vendas
/vendas/:CLIENTE/alterar	PUT	Altera cadastro de vendas

3.9 Gerenciamento de configuração e mudanças

Na fase de desenvolvimento da API, proposta por este estudo, utilizou-se IDE de desenvolvimento o *Visual Studio Code*, integrado com a ferramenta GitLab, utilizada para realizar o acompanhamento das funcionalidades implementadas de maneira dinâmica e intuitiva, e principalmente seu controle de versionamentos dos arquivos, o que torna o desenvolvimento mais produtivo e seguro, evitando perda de códigos. Essa ferramenta mantém o histórico de todo processo.

Na ferramenta foi criado o projeto chamado API integração PDV, podendo ser clonado através do link: https://gitlab.com/rodrigofelippeac/api-integracao-pdv.git, a figura 15 é uma captura da tela do projeto, onde é possível visualizar o projeto e seus arquivos.

Figura 15 – Projeto no GitLab



Fonte: Captura de tela site: http://gitlab.com

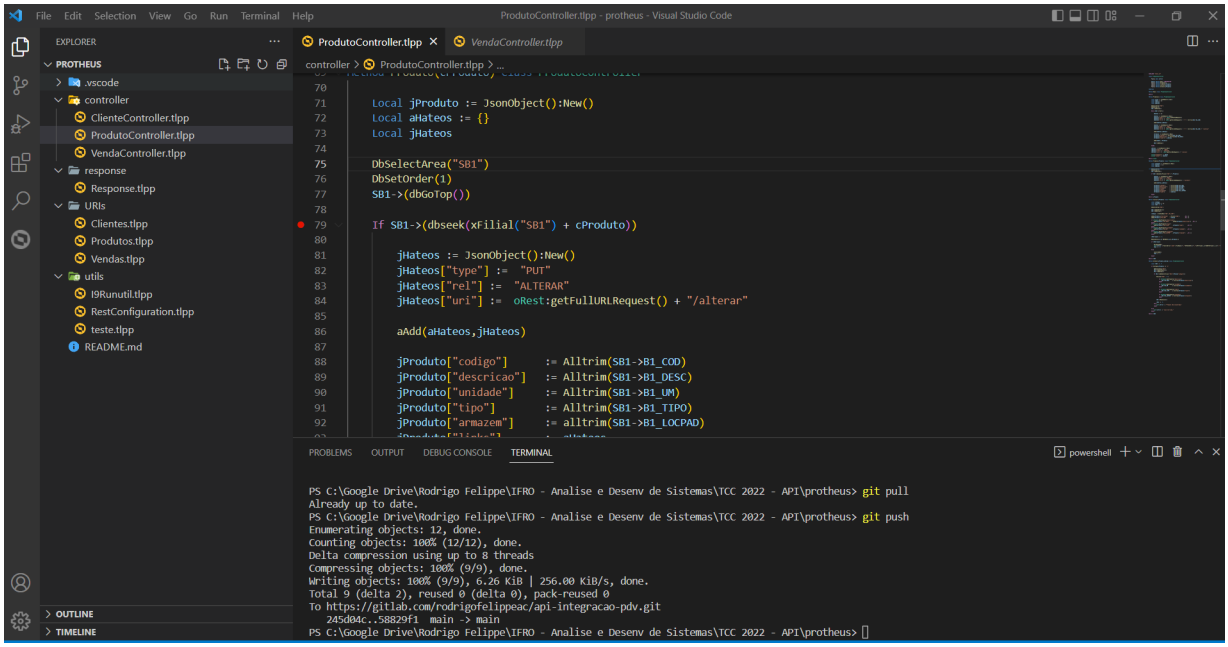
A ferramenta acima citada, também possui algumas opções de controle de projeto com metodologias ágeis, como por exemplo o Kanban, porém, nesse trabalho apenas utilizamos seu controle de versionamento.

3.10 Processo de desenvolvimento

Para desenvolvimento dessas funcionalidades propostas na API, foi utilizado a ferramenta de desenvolvimento *Visual Studio Code* que é um editor de código-fonte desenvolvido pela Microsoft, essa ferramenta suporta inúmeras linguagem, e sendo assim existe um plugin desenvolvido pela TOTVS, para conexão com ser servidores que rodam um serviço da camada de aplicação. Esse plugin permite a compilação dos programas desenvolvidos sejam eles em no formado de da linguagem de programação ADVPL ".prw"ou no formato na nova versão do ADVPL a linguagem TL++, que são os arquivos com a extenção ".tlpp".

A estrutura do projeto foi dividido em 4 pastas, sendo elas *controllers*, *responses* e URI´s visível na figura 16. Essa figura apresenta a arvore dos arquivos dentro da IDE de desenvolvimento.

Figura 16 – Projeto no *Visual Studio Code*



Fonte: Captura de tela *Visual Studio Code*

O sistema Protheus é um sistema ERP bastante completo e complexo, e cada empresa que utiliza esse software possui uma maneira de trabalhar com cada etapa dos processos. Sendo assim o sistema Protheus possui uma infinidade de maneiras para controlar seu fluxo de processos e validações de dados. Tendo em mente essas premissas, a empresa desenvolvedora disponibiliza algumas rotinas automáticas. Essas rotinas auxiliam o desenvolvedor a inserir dados dentro do sistema de maneira segura, fazendo essa inserção apenas possível desde que todas as validações do sistema sejam cumpridas. O desenvolvimento dessa API utiliza essas rotinas automáticas para realização de suas inclusões e alterações em seu banco de dados.

Podemos ver um exemplo de rotina automática para inclusão de produtos disponibilizada através do site da desenvolvedora TOTVS (TOTVS, 2022).

```
#INCLUDE "RWMAKE.CH"
#INCLUDE "TBICONN.CH"

User Function TMata010()
    Local aVetor := {}
    private lMsErroAuto := .F.

    PREPARE ENVIRONMENT EMPRESA "99" FILIAL "01" MODULO "EST"

    aVetor:= { {"B1_COD" ,"9994" ,NIL},;
     {"B1_DESC" ,"PRODUTO TESTE - ROTINA AUTOMATICA" ,NIL},;
     {"B1_TIPO" ,"PA" ,Nil},;
     {"B1_UM" ,"UN" ,Nil},;
     {"B1_LOCPAD" ,"01" ,Nil},;
     {"B1_PICM" ,0 ,Nil},;
     {"B1_IPI" ,0 ,Nil},;
     {"B1_CONTRAT" ,"N" ,Nil},;
     {"B1_LOCALIZ" ,"N" ,Nil}}

    MSExecAuto({|x,y| Mata010(x,y)},aVetor,3)

    aVetor:= { {"B1_COD" ,"9994" ,NIL},;
     {"B1_DESC" ,"PRODUTO TESTE - ALTERADO" ,NIL}}

    MSExecAuto({|x,y| Mata010(x,y)},aVetor,4)

    aVetor:= { {"B1_COD" ,"9994" ,NIL},;
     {"B1_DESC" ,"PRODUTO TESTE - ROTINA AUTOMATICA" ,NIL}}

    MSExecAuto({|x,y| Mata010(x,y)},aVetor,5)

    If lMsErroAuto
     MostraErro()
    Else
     Alert("Alteracao realizada com sucesso")
    Endif

Return
```

Quadro 3.1 – Exemplo de função utilizando execauto.

Na linha 1 e 2, são adicionado as includes da TOTVS, necessários para compilação do código. Na linha 9 é possível verificar a abertura da empresa, nesse momento o sistema Protheus identifica qual empresa está sendo utilizada, e podemos verificar na linha 21 o uso correto dessa rotina automática.

Utilizando-se dessas rotinas disponíveis foram criadas classes responsáveis pro controlar a consulta, inserção e alteração de dados no sistemas. Um exemplo de classe criada para a funcionalidade de inclusão de clientes, a *ClienteController.tlpp*, recebe os dados através do serviço *web* e faz a inclusão do cliente no sistema Protheus:

[]

```
Method Incluir(jCliente) Class ClienteController

    Local cCodigo
    Local aCliente := {}
    Local lRet := .F.

    dbSelectArea("SA1")
    SA1->(dbSetOrder(1))
    SA1->(dbGoTop())

    cCodigo := GetSxeNum("SA1","A1_COD")

    aCliente := {}

    aAdd(aCliente,{"A1_FILIAL"   , xFilial("SA1")       ,Nil })
    aAdd(aCliente,{"A1_COD"     , cCodigo         ,Nil })
    aAdd(aCliente,{"A1_LOJA"    ,"01"            ,NIL})
    aAdd(aCliente,{"A1_RISCO"    ,"A"              ,NIL})

    If jCliente:HasProperty("pessoa")
        aAdd(aCliente,{"A1_PESSOA"   ,UPPER(jCliente["pessoa"])     ,NIL})
    EndIf
    If jCliente:HasProperty("pessoa")
        aAdd(aCliente,{"A1_NOME"    ,UPPER(jCliente["nome"])       ,NIL})
        aAdd(aCliente,{"A1_NREDUZ"   ,UPPER(jCliente["nome"])      ,NIL})
    EndIf
    If jCliente:HasProperty("cgc")
        aAdd(aCliente,{"A1_CGC"      ,jCliente["cgc"]        ,NIL})
    EndIf
    If jCliente:HasProperty("endereco")
        aAdd(aCliente,{"A1_END"     ,UPPER(jCliente["endereco"])     ,NIL})
    EndIf
    If jCliente:HasProperty("tipo")
        aAdd(aCliente,{"A1_TIPO"     ,jCliente["tipo"]     ,NIL})
    EndIf
    If jCliente:HasProperty("estado")
        aAdd(aCliente,{"A1_EST"     ,UPPER(jCliente["estado"])     ,NIL})
    EndIf
    If jCliente:HasProperty("municipio")
        aAdd(aCliente,{"A1_MUN"      ,jCliente["municipio"]    ,NIL})
```

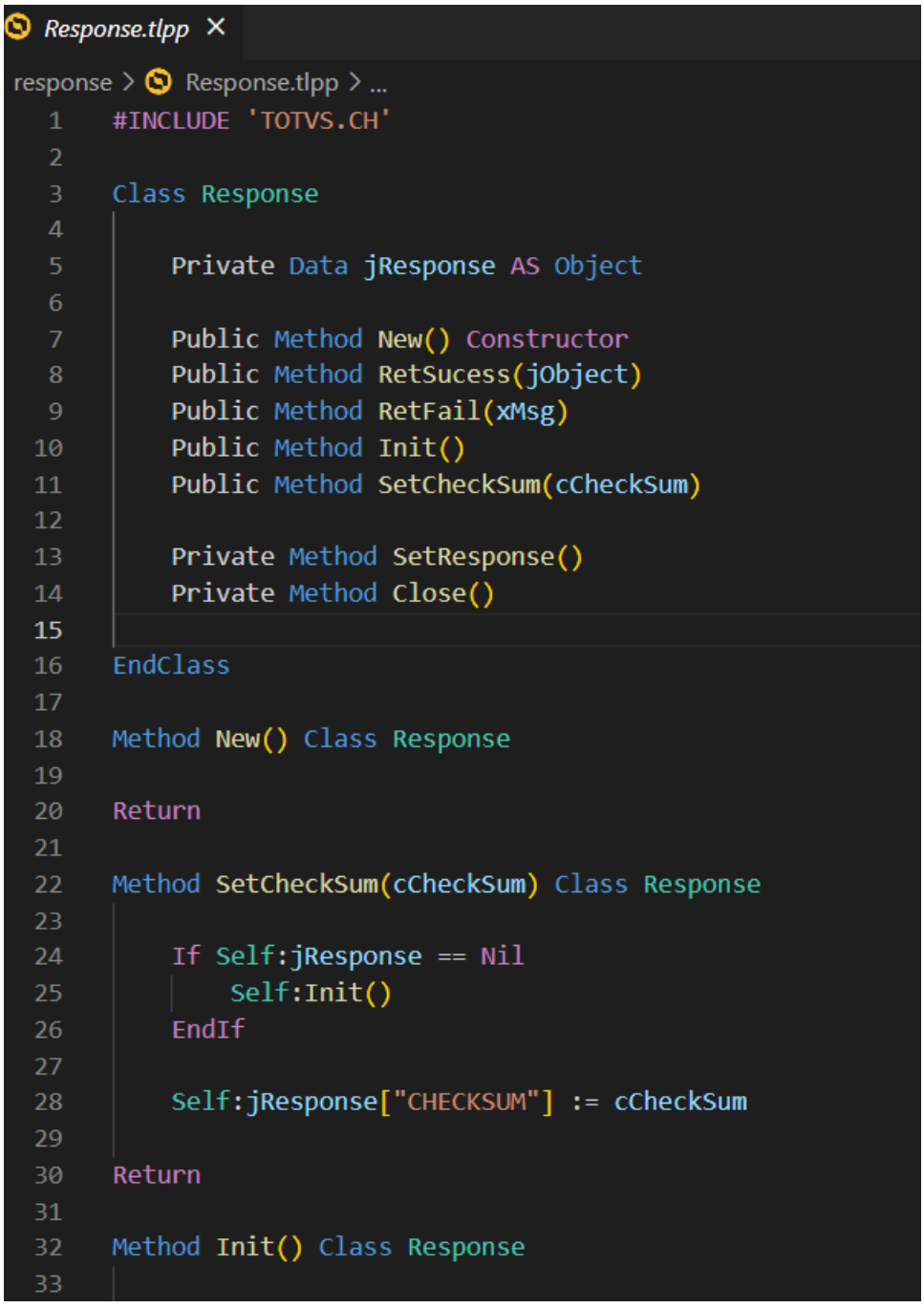
```
    EndIf
    If jCliente:HasProperty("bairro")
        aAdd(aCliente,{"A1_BAIRRO"   ,jCliente["bairro"]    ,NIL})
    EndIf
    If jCliente:HasProperty("cep")
        aAdd(aCliente,{"A1_CEP"      ,jCliente["cep"]       ,NIL})
    EndIf
    If jCliente:HasProperty("ddd")
        aAdd(aCliente,{"A1_DDD"      ,jCliente["ddd"]       ,NIL})
    EndIf
    If jCliente:HasProperty("telefone")
        aAdd(aCliente,{"A1_TEL"      ,jCliente["telefone"]    ,NIL})
    EndIf

    If jCliente:HasProperty("email")
        aAdd(aCliente,{"A1_EMAIL"    ,jCliente["email"]       ,NIL})
    EndIf

    lMsErroAuto := .F.

    MSExecAuto({|x,y| MATA030(x,y)},aCliente,3)

    If lMsErroAuto

        RollBackSX8()
        self:cError := MostraErro("\LOG\",ProcName()+"_"+DTOS(DATE())+"_"+
            LEFT(Time(),2)+SUBSTR(Time(),4,2)+"_"+STRZERO(SECONDS(),10)+".
            LOG")
        lRet := .F.

    Else

        ConfirmSX8()
        lRet := .T.

    EndIf

Return lRet
```

Quadro 3.2 – Exemplo de função utilizando execauto.

No código acima, é possível verificar que o método faz Incluir, realiza a abertura do base de dados utilizando-se da função "DbSelectArea"na linha 8, o código identificador do cliente também é requisitado através de outra função controladora de numerações nativa do sistema Getsxenum(), é montado um vetor com todas as informações necessários para o cadastro de um cliente, e esse é feito através da rotina automática "MATA030", como podemos observar na linha 62 desse código.

Uma necessidade de resposta padrão do *webservice* foi identificado, para que seja possível utilizar uma interface sempre quem um novo *endpoint* foi criado, com essa premissa foi desenvolvida a classe *Response.tlpp*, que é implementada sempre que um novo conjunto de *endpoints* necessite ser desenvolvido, essa classe é responsável pelas estrutura das respostas, tornando assim simplificado e padronizado o sistema de resposta da API.

Figura 17 – *Visual Studio Code* - Response.tlpp



Fonte: Captura de tela Visual Studio Code

Nessa classe foi implementado os métodos RetSucces() e RestFail() que são responsáveis pelas respostas, seja ela uma resposta de sucesso, ou de falha. Criando assim um padrão para todas as respostas através das classes que implementam essa classe.

4 Resultados e discussões

Neste capítulo são apresentados os resultados e discussões acerca da API proposta nesse trabalho.

4.1 Testes e Validação dos *Endpoints*

Nesta seção, serão exibidos os aspectos relevantes utilizados para a implementação da API INTERAÇÃO PDV, assim como as tecnologias e funcionalidades com os princípios REST, nesse momento é possível visualizar o resultado de testes realizados para validações dos *endpoints* anteriormente definidos. Os *endpoints* são os pontos de acesso disponibilizados de uma aplicação para que o cliente utilize para consumir os serviços providos por uma API REST, sendo classificadas em recursos como os principais tópicos disponibilizados. Assim, segundo Fielding e Taylor (2000) quando os recursos são disponibilizados por (URI) fazem parte de uma restrição da interface uniforme propostas pelo mesmo, os quais servem como ponto de entrada para o consumo da API.

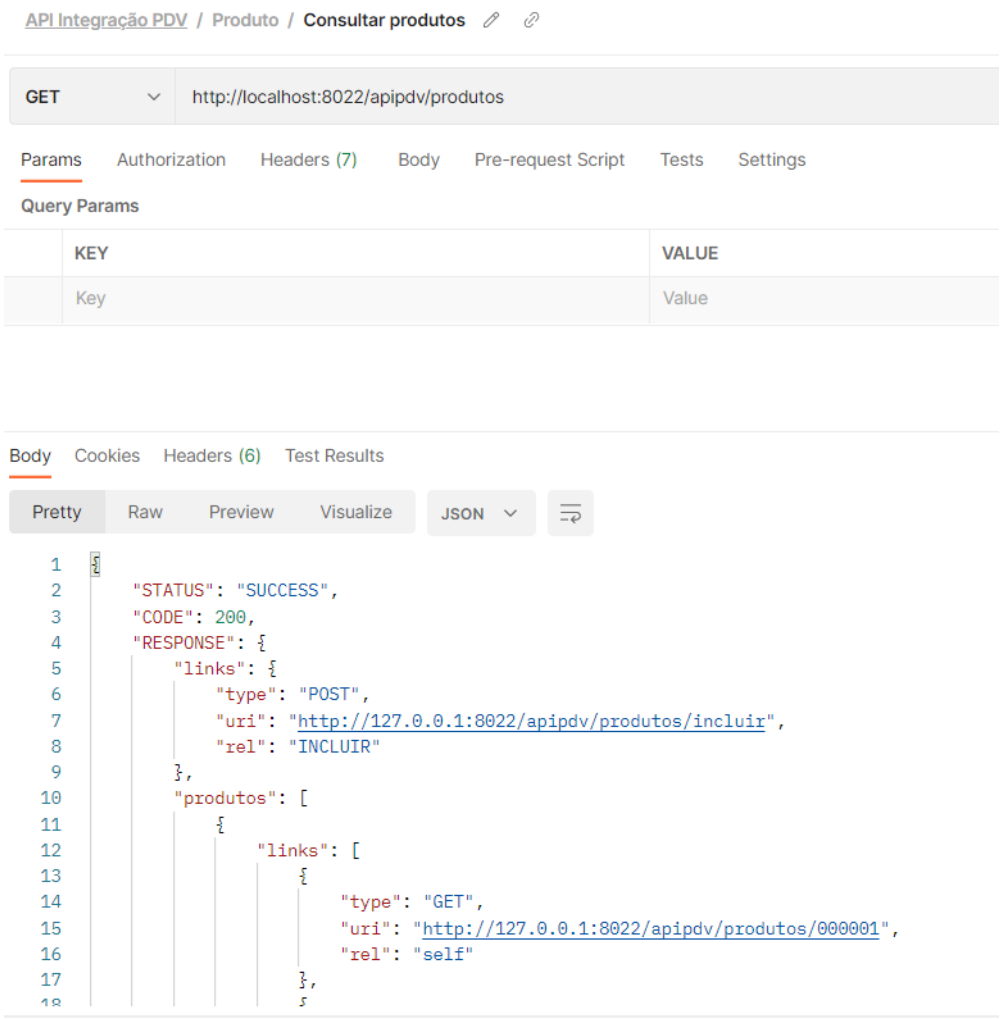
Esses *endpoints* foram divididos em 3 categorias: Produtos, Clientes e Vendas.

Com a aplicação executada, pode-se realizar requisições, através da ferramenta Postman, para verificar o comportamento da API em cada *endpoint*, o resultado é exibido abaixo de acordo com suas categorias.

4.1.1 Produto

- **Lista Produto** - Assim, ao realizar a requisição através da rota "/produtos", utilizando o método GET obteve-se a lista de todos os produtos, como resposta um JSON estruturado conforme figura 18.

Figura 18 – Captura de tela POSTMAN - Lista Produto



Fonte: captura realizada pelo autor

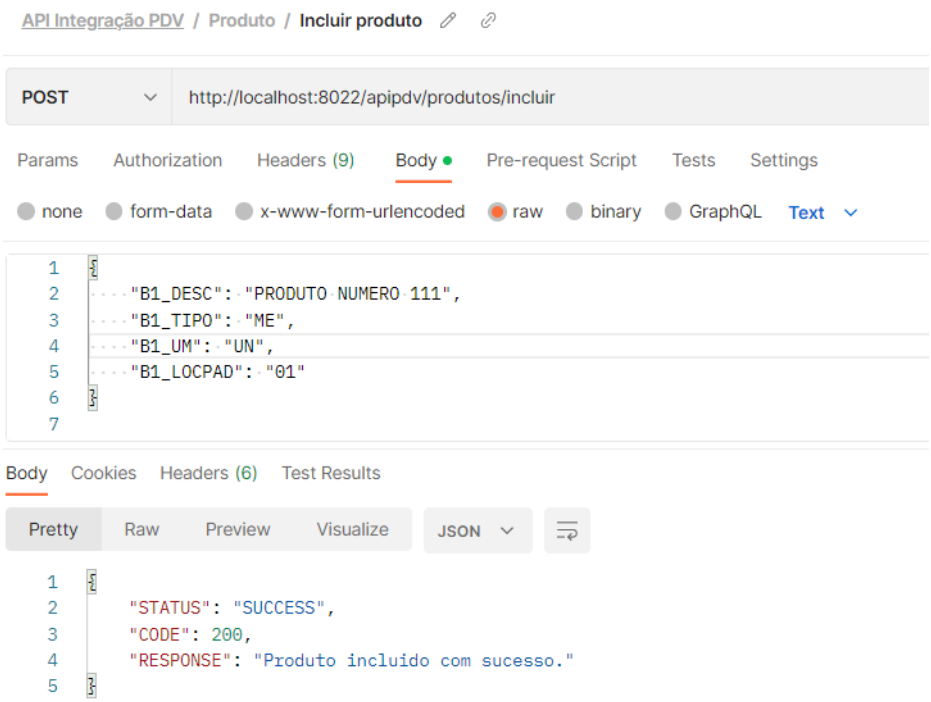
- **Incluir Produto** – A inclusão de produto é feita através do endereço "/produtos/incluir". Para a inclusão desse produto, é feito uma requisição POST, informando os seguintes dados obrigatoriamente conforme código:

```
{
    "descricao": "PRODUTO NUMERO 2",
    "tipo": "ME",
    "unidade": "UN",
    "armazem": "01"
}
```

Quadro 4.1 – Estrutura JSON para inclusão de produto.

E o resultado é apresentado da seguinte forma no POSTMAN conforme figura 19:

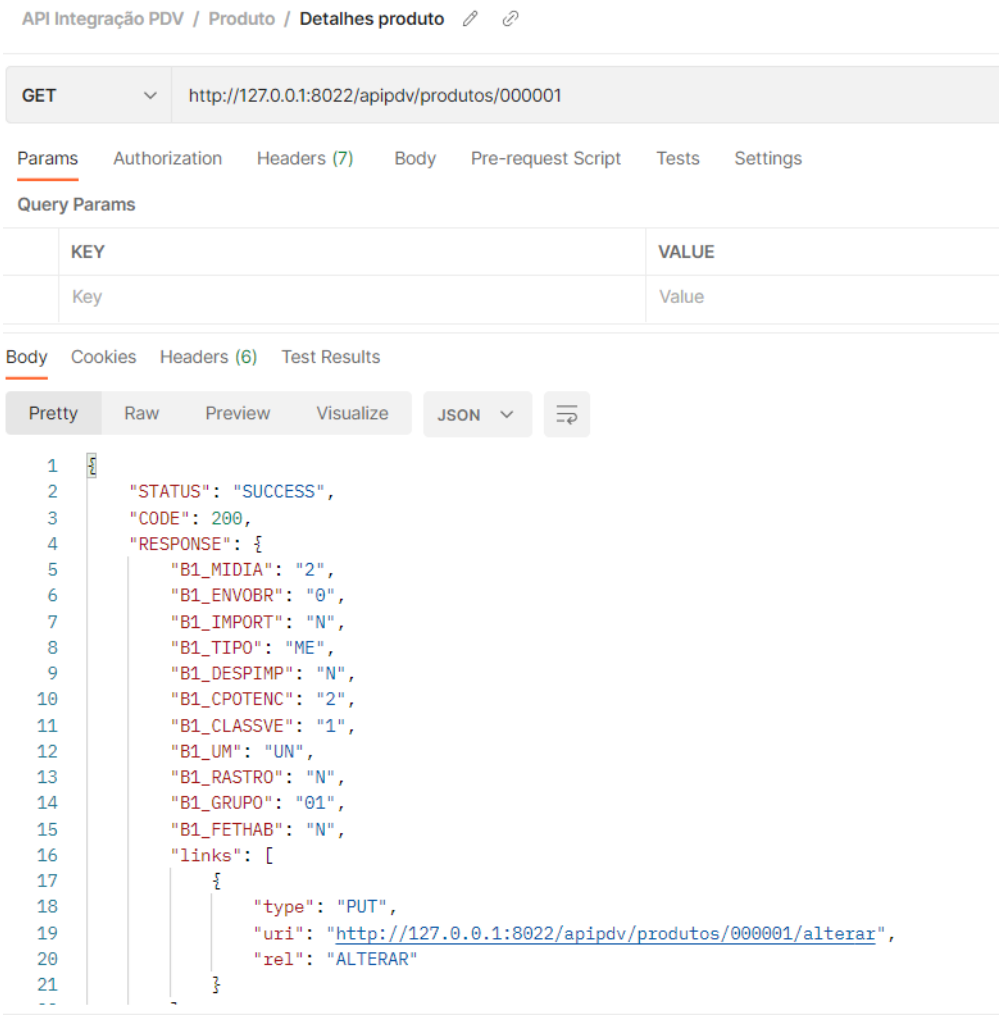
Figura 19 – Captura de tela POSTMAN - Incluir Produto



Fonte: captura realizada pelo autor

- **Detalha Produto** – Para se obter os detalhes de cada produto foi disponibilizado através do URI "/produtos/:PRODUTO" onde o ":PRODUTO" representa o código do produto a ser consultado, e a figura 20 apresenta o seu resultado:

Figura 20 – Captura de tela POSTMAN Detalha Produto



Fonte: captura realizada pelo autor

4.1.2 Cliente

Clientes são os principais atores na realização dos atendimentos, sendo assim, clientes novos são comuns no dia a dia do trabalho, e com essa informação definimos algumas URI´s para cadastro e consulta de clientes. Essas consultas são as necessárias para se manter um cadastro sempre atualizado, a manutenção desse cadastro consiste em Listar, Consultar, Incluir e Alterar.

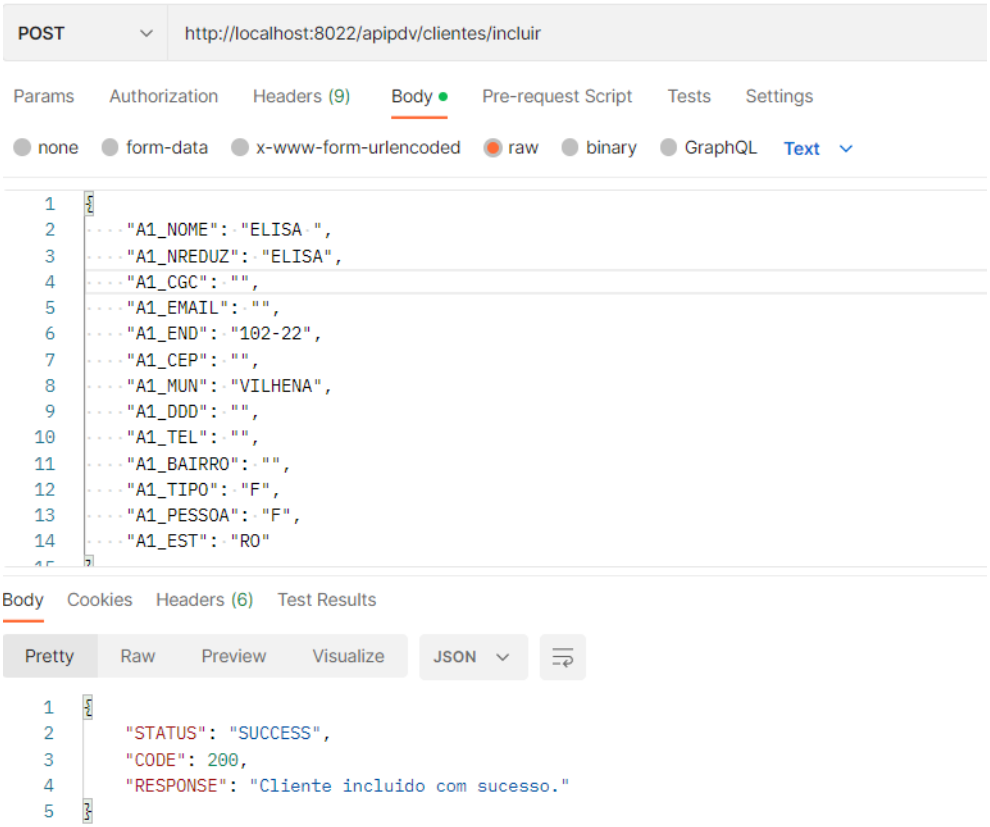
- **Incluir Cliente** – A inclusão de cliente, deve ser realizado seguindo as obrigatoriedades do sistema ERP. Abaixo o modelo definido de JSON para realização do cadastro, que precisa ser realizado através do método POST na URI "/cliente/incluir", como no exemplo a seguir:

```
{
  "nome": "RODRIGO FELIPPE ALVES",
  "cgc": "99999999999",
  "email": "rodrigo@gmail.com",
  "endereco": "102-22",
  "cep": "76980000",
  "municipio": "VILHENA",
  "ddd": "69",
  "telefone": "981336194",
  "bairro": "BAIRRO",
  "tipo": "F",
  "pessoa": "F",
  "estado": "RO"
}
```

Quadro 4.2 – Estrutura JSON para inclusão de Cliente.

Em caso de sucesso obtem-se o seguinte resultado visualizado na figura 21:

Figura 21 – Captura de tela POSTMAN Incluir Cliente



Fonte: captura realizada pelo autor

- **Listar e consultar cliente** – Essas formas seguem o mesmo princípio, no URI "/clientes/" é exibida uma lista de todos os clientes disponíveis, e o *endpoint* "/cliente/:CLIENTE" onde o ":CLIENTE" é a variável referente ao código do cliente no sistema ERP

4.1.3 Vendas

Os *endpoints* de vendas são os principais *endpoints* dessa API, é nele que vendas são enviadas através de qualquer aplicação para serem efetivadas no sistema de ERP, essa venda precisa de informações referentes a clientes, produtos, quantidade, formas de pagamento e código do vendedor. Os dados do cliente precisam ser enviados para que seja possível realizar a venda no nome dele, também é enviado uma lista de produtos que nesse caso definimos como "itens" informando código do produto, a quantidade, seu preço unitário e o valor total, e uma lista com as formas de pagamento, inicialmente apenas implementamos o recebimento a vista.

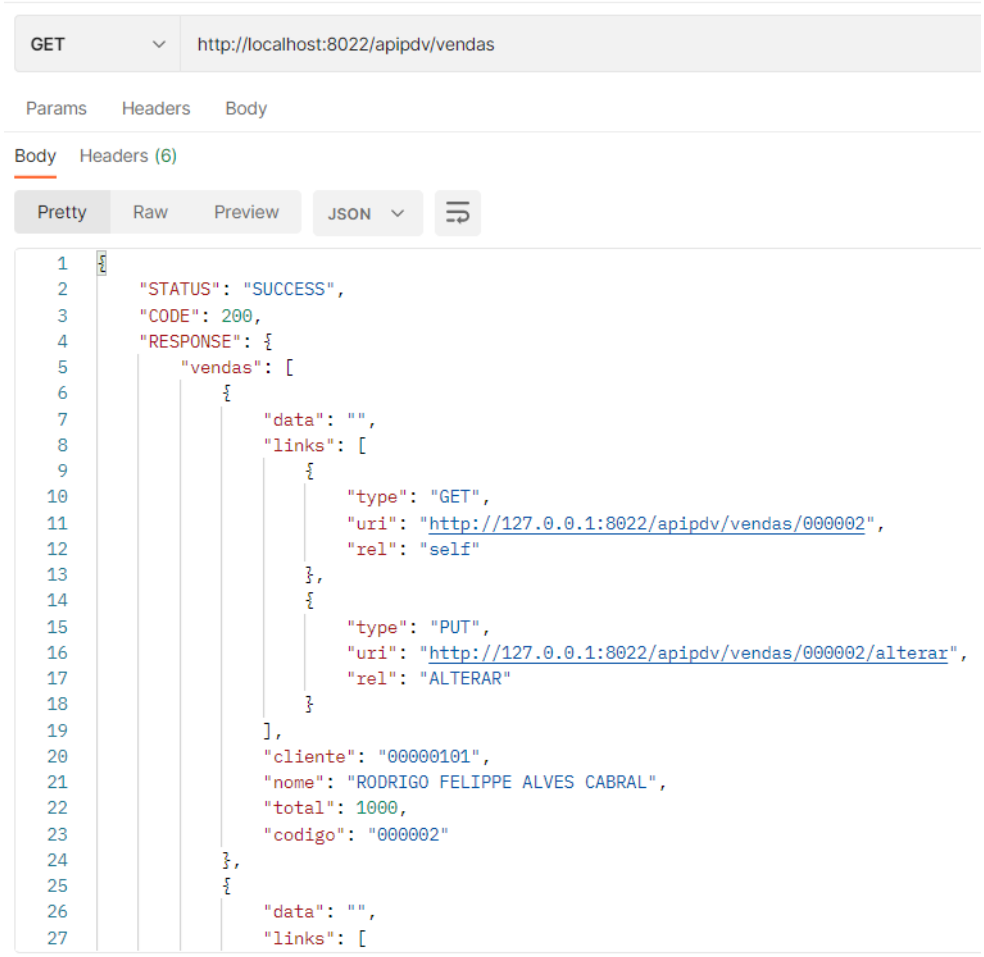
- **Incluir Venda** – A inclusão da venda é feita através da URI "/vendas/incluir", devendo seguir o modelo de arquivo JSON abaixo, sendo enviado através do *header* do método http POST:

```
{
  "cliente": {
    "cgc": "04881411411",
    "codigo": "00000101",
    "email": "rodrigo@i9on.com.br",
    "nome": "RODRIGO FELIPPE ALVES CABRAL"
  },
  "data": "2021-06-15 11:19:15.053",
  "formasdepagamento": [
    {
      "descricao": "Dinheiro",
      "sigla": "R$",
      "valor": 1000
    }
  ],
  "idprotheus": "01000001",
  "itens": [
    {
      "codigo": "000001",
      "descricao": "R90 30MB FILTRO RACOR",
      "preco":   1000,
      "quantidade": 1,
      "total": 1000
    }
  ],
  "status": "Finalizado",
  "total": 1000,
  "vendedor": "000001"
}
```

Quadro 4.3 – Estrutura JSON para inclusão de Cliente.

- **Consulta vendas** – É possível obter uma lista das vendas ao consultar as vendas realizadas no sistema ERP através da URI "/vendas" e seu resultado pode ser verificado na figura 22 abaixo:

Figura 22 – Captura de tela POSTMAN Consulta vendas

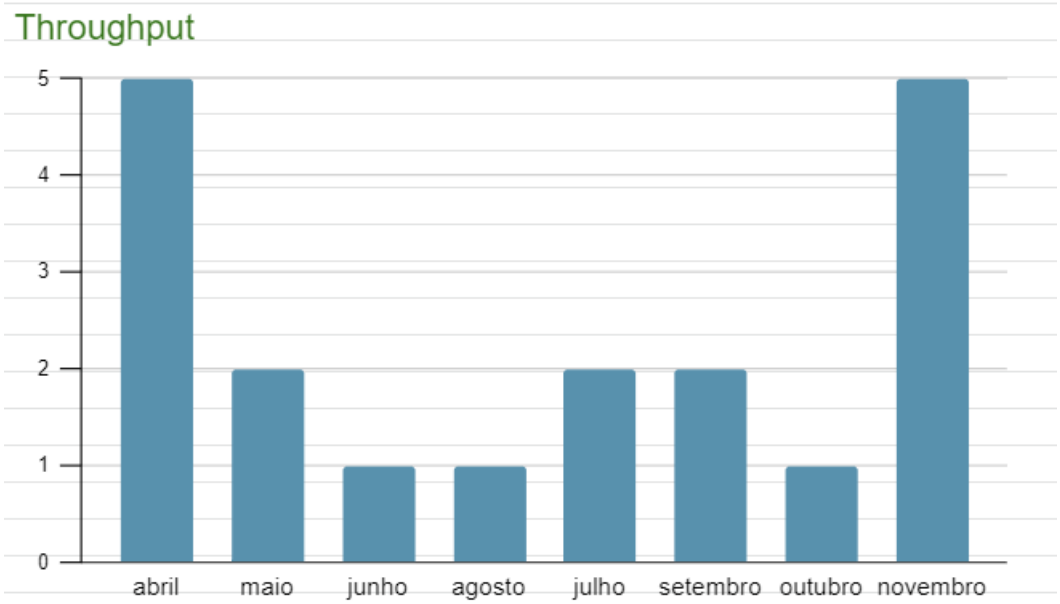


Fonte: captura realizada pelo autor

4.2 Métricas

A métrica utilizada no nosso modelo de gerenciamento de atividades desenvolvido foi o *Throughput*, que é a quantidade de vazão do sistema, ou seja, quantas tarefas foram entregues num determinado período. É uma métrica que mede a velocidade na qual uma certa quantidade de cards em um quadro Kanban são entregues num determinado período de tempo como explica (CAMARGO ROBSON E RIBAS, 2019). É possível visualizar o resultado das atividades realizadas no período de abril a novembro, na figura 23.

Figura 23 – Gráfico Throughput



Fonte: captura realizada pelo autor

4.3 Documentação

A documentação foi desenvolvida com auxílio da ferramenta Postman, essa ferramenta auxilia na documentação dos *endpoints*, essa documentação é realizada a medida que os testes estão sendo feitos e os resultados dos testes já podem virar um exemplo da documentação. Na documentação que está disponível no endereço:

https://documenter.getpostman.com/view/12490767/2s7YYvZh61

Quadro 4.4 – Endereço da documentação.

É possível verificar os métodos disponíveis, juntamente com os exemplos de em respostas em método *GET* e em casos que o método de envio é *POST*. possui exemplo de corpo a ser enviado.

A figura 24 apresenta a tela principal da documentação, que pode ser acessado por desenvolvedores, possibilitando o desenvolvimento de ferramentas para consumo dos serviços disponíveis.

Figura 24 – Documentação API Postman

Fonte: captura realizada pelo autor

4.4 Implementação

A implementação dessa API necessita a configuração do *Webservices* do ERP Protheus e a aplicação dos arquivos fontes criados a partir desse trabalho. Com a aplicação desses fontes, os *endpoints* começaram a responder as requisições.

5 Considerações finais

O entendimento do processo de pré-venda e orçamentos juntos aos usuários do sistema e a experiência de anos trabalhando com esse sistema ERP Protheus, resultou em uma API REST com as premissas dos requisitos idealizados para a API, baseando-se nos padrões definidos no estilo arquitetural REST. O trabalho permitiu a criação de uma API simples e fácil implementação, em que o desenvolvedor consegue realizar vendas no sistema ERP Protheus sem a necessidade de conhecer o código fonte ou o sistema, podendo consumi-los através das requisições para uma URI (*EndPoint*) HTTP. Com a padronização dos recursos cria-se um potencial para interoperabilidade, ou seja, comunicar-se com outras aplicações em plataformas distintas, bem como a implementação de novos serviços, gerando assim novas versões. Acima de tudo a implementação respeita todas as regras de negócios aplicadas no Sistema ERP Protheus, não sendo possível a interação com sistema sem que as validações existentes sejam contempladas, isso centraliza de forma a manter a integridade das aplicações. Foi identificado uma melhora imensa nas possibilidades de integrações com outros softwares, seja qual for a linguagem.

5.1 Trabalhos futuros

Após o desenvolvimento dessa API foi identificado uma serie de outros *endpoints* que podem ser adicionados para complementar essa API. Também é importante a criação de alguns relatórios de resultados para que seja possível o acesso a informações no sistema que só são possíveis no momento através do ERP, então a ideia é desenvolver uma *framework* para criação de uma central de relatórios no sistema, sem a necessidade de uma nova implementação a cada relatório. Com isso o principal trabalho futuro é o desenvolvimento de uma aplicação móvel Android para consumir essa API, realizando todas as atividades de vendas e consultas.

- Manter Vendedores
- Manter formas de pagamento
- Cancelamento de Notas Fiscal
- XML de Nota Fiscal
- Central de Relatórios
- Aplicativo Mobile em Android para consumir essa API

Referências

CAMARGO ROBSON E RIBAS, T. *Gestão ágil de projetos, as melhores soluções para suas necessidades. 1º Edição.* [S.l.]: Saraiva, 2019. Citado 2 vezes nas páginas 19 e 55.

DAVENPORT, T. H. *Ecologia da Informação: porque só a tecnologia não basta para o sucesso na era da informação. 2ª ed.* [S.l.]: Futura, 1998. Citado na página 20.

GITLAB. *What is version control?* 2022. Acessado em: 11 de setembro de 2022. Disponível em: <https://about.gitlab.com/topics/version-control/>. Citado na página 19.

GOSALA, B. *Automatic Classification of UML Class Diagrams Using Deep Learning Technique: Convolutional Neural Network. Applied Sciences, Basel.* [S.l.]: Applied Sciences, Basel, 2021. Citado na página 18.

GUEDES, G. T. A. *UML 2: uma abordagem prática. 2ª. ed.* [S.l.]: Novatec Editora, 2011. Citado na página 18.

HEFLO. *Ferramenta de BPM Completa.* 2022. Acesso em: 25 de maio de 2022. Disponível em: <https://www.taiga.io/>. Citado na página 18.

JUNIOR, C. C. *Sistemas integrados de gestão: ERP - uma abordagem gerencial. 1. Ed.* [S.l.]: Intersaberes, 2012. Citado na página 20.

MARQUES, A. I. A. *Desenvolvimento de API para aplicação cloud.* [S.l.]: Tese de Doutorado, 2018. Citado na página 29.

MUSSER, J. *Open apis: state of the market. In: The Glue Conference.* [S.l.]: S.n., 2011. Citado na página 29.

O'GRAND, A. *Gitlab Quick Start Guide.* [S.l.]: Packt, Birmingham - Mumbai, 2018. Citado na página 19.

PORTALERP. *Maior site sobre Software e Gestão.* 2022. Acesso em: 20 de junho de 2022. Disponível em: <https://portalerp.com/>. Citado na página 20.

POSTMAN. *Postman API Plataform.* 2022. Acesso em: 02 de outubro de 2022. Disponível em: <https://www.postman.com/>. Citado na página 31.

REDHAT. *O que é API?* 2022. Acesso em: 12 de outubro de 2022. Disponível em: <https://www.devmedia.com.br/gerencia-de-configuracao-e-mudancas/31327>. Citado na página 29.

ROSS, K. W. KUROSE J. F. e. *Redes de computadores e a internet: uma abordagem top-down. 6 Edição.* [S.l.]: Pearson Prentice-Hall, 2013. Citado na página 26.

SAUDATE, A. *REST. Construa API's inteligentes de maneira simples, 1ª Edição.* [S.l.]: Casa do Código, 2013. Citado 3 vezes nas páginas 16, 29 e 30.

SHIH Y. E HUANG, S. *The actual usage of ERP systems: An extended technology acceptance perspective. Journal of Research and Practice in Information Technology.* [S.l.]: Ballarat, 2009. Citado na página 15.

TAIGA. *Your Opensource agile project management software*. 2022. Acesso em: 25 de agosto de 2022. Disponível em: <https://www.taiga.io/>. Citado na página 19.

TANEBAUM, A. S. *Redes de computadores. (Tradução: Vandenberg D. de Souza). 4º Edição*. [S.l.]: R. Amsterdam, 2010. Citado 3 vezes nas páginas 18, 26 e 27.

TOTVS. *Conheça todos os detalhes do sistema Protheus*. 2022. Acesso em: 10 de abril de 2022. Disponível em: <https://www.totvs.com/blog/erp/sistema-protheus/>. Citado 6 vezes nas páginas 6, 7, 21, 24, 25 e 42.

WEILL P.; ROSS, J. *Governança de TI: como as empresas com melhor desempenho administram os direitos decisórios de TI na busca por resultados superiores*. [S.l.]: Makron Books do Brasil Editora Ltda, 2006. Citado na página 15.

APÊNDICE A – Documentação da API

Documentation Settings

API Integração PDV

API de integração com PDV, essa API é composta de um conjunto de metodos para realização de cadastro de produto, clientes e cadastro de vendas.

Produto

GET Consultar produtos

http://localhost:8022/apipdv/produtos

Método para obter todos os produtos.

Retorna lista de produtos.

Example Request Consultar produtos

```
curl --location --request GET 'http://localhost:8022/apipdv/produtos'
```

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "links": {
      "type": "POST",
      "uri": "http://127.0.0.1:8022/apipdv/produtos/incluir",
      "rel": "INCLUIR"
    }
```


View More

Documentation Settings

http://127.0.0.1:8022/apipdv/produtos/000001

Método para obter detalhes do produto.

Example Request Detalhes produto

```
curl --location --request GET 'http://127.0.0.1:8022/apipdv/produtos/000001' \
--data-raw ''
```

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "B1_MIDIA": "2",
    "B1_ENVOBR": "0",
    "B1_IMPORT": "N",
    "B1_TIPO": "ME",
```


View More

POST Incluir produto

http://localhost:8022/apipdv/produtos/incluir

Método para cadastro de produto

BODY raw


```
{
    "B1_DESC": "PRODUTO NUMERO 111",
    "B1_TIPO": "ME",
    "B1_UM": "UN",
    "B1_LOCPAD": "01"
}
```

Documentation Settings

```
curl --location --request POST 'http://localhost:8022/apipdv/produtos/incluir' \
--data-raw '{
    "B1_DESC": "PRODUTO NUMERO 03",
    "B1_TIPO": "ME",
    "B1_UM": "UN",
    "B1_LOCPAD": "01"
}
'
```

Example Response 200 OK

Body Header (6)

```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": "Produto incluido com sucesso."
}
```

PUT Alterar Produto

http://127.0.0.1:8022/apipdv/produtos/000001/alterar

BODY raw

```
{
    "B1_DESC": "PRODUTO NUMERO 01",
    "B1_TIPO": "ME",
    "B1_UM": "UN",
    "B1_LOCPAD": "01",
    "B1_COD": "UN"
}
```

Example Request Alterar Produto

```
curl --location --request PUT 'http://127.0.0.1:8022/apipdv/produtos/000001/alterar' \
--data-raw '{
    "B1_DESC": "PRODUTO NUMERO 01",
    "B1_TIPO": "ME",
    "B1_UM": "UN",
    "B1_LOCPAD": "01",
```

Documentation Settings

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": "Produto alterado com sucesso."
}
```


Cliente

GET Consultar clientes

http://localhost:8022/apipdv/clientes

Método para obter todos os produtos.

Example Request Consultar clientes

```
curl --location --request GET 'http://localhost:8022/apipdv/clientes'
```

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "clientes": [
      {
        "links": [
          {
```


View More

Documentation Settings

http://localhost:8022/apipdv/clientes/00000101

Example Request Detalhes cliente

```
curl --location --request GET 'http://localhost:8022/apipdv/clientes/00000101'
```

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "A1_PESSOA": "F",
    "A1_MOEDALC": 2,
    "A1_EST": "RO",
    "A1_RECCSLL": "N",
```


View More

POST Incluir cliente

http://localhost:8022/apipdv/clientes/incluir

Método para cadastro de produto

BODY raw


```
{
    "A1_NOME": "ELISA ",
    "A1_NREDUZ": "ELISA",
    "A1_CGC": "",
    "A1_EMAIL": "",
    "A1_END": "102-22",
    "A1_CEP": "",
    "A1_MUN": "VILHENA",
```


View More

Documentation Settings

Example Request — Incluir cliente

```
curl --location --request POST 'http://localhost:8022/apipdv/clientes/incluir' \
--data-raw '{
    "A1_NOME": "GABRIELE MEDEIROS ALVES",
    "A1_NREDUZ": "GABRIELE",
    "A1_CGC": "",
    "A1_EMAIL": "",
    "A1_END": "102-22",
    "A1_CEP": "",
```

View More

Example Response — 200 OK

Body Header (6)

```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": "Cliente incluido com sucesso."
}
```

PUT Alterar cliente

http://127.0.0.1:8022/apipdv/clientes/00000701/alterar

Método para alteração do cadastro de produto.

BODY raw

```
{
    "A1_NOME": "WILLIAM SILVA SANTOS",
    "A1_NREDUZ": "WILLL",
    "A1_CGC": "",
    "A1_EMAIL": "",
    "A1_END": "102-22",
    "A1_CEP": "",
    "A1_MUN": "VILHENA",
```

View More

Documentation Settings

```
curl --location --request PUT 'http://127.0.0.1:8022/apipdv/clientes/00000101/alterar' \
--data-raw '{
    "codigo": "00000101",
    "nome": "RODRIGO FELIPPE ALVES CABRAL",
    "cgc": "",
    "email": "rodrigofelippeac@gmail.com",
    "endereco": "102-22",
    "cep": "",
```

View More

Example Response 200 OK

Body Header (6)

```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": "Cliente alterado com sucesso."
}
```

Vendedor

GET Consultar venderores

http://localhost:8022/apipdv/vendedores

Método para obter todos os produtos.

Example Request Consultar venderores

```
curl --location --request GET 'http://localhost:8022/apipdv/vendedores'
```

Formas

Documentation Settings

GET Consultar Formas

http://localhost:8022/apipdv/formas

Método para obter todos os produtos.

Example Request Consultar Formas

```
curl --location --request GET 'http://localhost:8022/apipdv/formas'
```

Venda

GET Consultar vendas

http://localhost:8022/apipdv/vendas

Método para obter todos os produtos.

Example Request Consultar vendas

```
curl --location --request GET 'http://localhost:8022/apipdv/vendas'
```

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "vendas": [
      {
```

Documentation Settings

GET Detalhes venda

http://127.0.0.1:8022/apipdv/vendas/000002

Example Request Detalhes venda

```
curl --location --request GET 'http://127.0.0.1:8022/apipdv/vendas/000002'
```

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "data": "",
    "links": [
      {
        "type": "PUT",
```


View More

POST Incluir venda

http://localhost:8022/apipdv/vendas/incluir

Método para cadastro de produto

BODY raw


```
{
  "cliente": {
    "cgc": "04881411411",
    "codigo": "00000101",
    "email": "rodrigo@i9on.com.br",
    "nome": "RODRIGO FELIPPE ALVES CABRAL"
```

Documentation Settings

Example Request Incluir venda

```
curl --location --request POST 'http://localhost:8022/apipdv/vendas/incluir' \
--data-raw '{
  "cliente": {
    "cgc": "04881411411",
    "codigo": "00000101",
    "email": "rodrigo@i9on.com.br",
    "nome": "RODRIGO FELIPPE ALVES CABRAL"
  },
```

View More

PUT Alterar venda

http://127.0.0.1:8022/apipdv/vendas/000001/alterar

Método para alteração do cadastro de produto.

BODY raw

```
{
    "codigo": "00000101",
    "nome": "RODRIGO FELIPPE ALVES CABRAL",
    "cgc": "",
    "email": "rodrigofelippeac@gmail.com",
    "endereco": "102-22",
    "cep": "",
    "municipio": "VILHENA",
```

View More

Example Request Alterar venda

```
curl --location --request PUT 'http://127.0.0.1:8022/apipdv/vendas/000001/alterar' \
--data-raw '{
    "codigo": "00000101",
    "nome": "RODRIGO FELIPPE ALVES CABRAL",
```

Documentation Settings


```
"cep": "",
```


View More

Utils

Tabela

GET Consultar tabelas

http://localhost:8022/apipdv/tabela/

Método para obter todos os produtos.

Retorna lista de produtos.

Example Request Consultar tabelas


```
curl --location --request GET 'http://localhost:8022/apipdv/tabela/'
```


Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "tabelas": [
      {
        "links": [
          {
```


View More

Documentation Settings

http://localhost:8022/apipdv/tabela/SB1

Example Request Consultar campos

```
curl --location --request GET 'http://localhost:8022/apipdv/tabela/SB1'
```

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "links": {
      "type": "GET",
      "uri": "http://127.0.0.1:8022/apipdv/tabela/SB1/obrigatorios",
      "rel": "Campos Obrigatorios"
    `
```


View More

GET Consultar Itens

No request URL found. It will show up here once added.

Example Request Consultar campos obrigatorios

```
curl --location --request GET 'http://localhost:8022/apipdv/tabela/SB1/obrigatorios'
```

Example Response 200 OK

Body Header (6)


```
{
  "STATUS": "SUCCESS",
  "CODE": 200,
  "RESPONSE": {
    "SB1": [
```

Documentation Settings

View More

Query

POST Query

http://localhost:8022/apipdv/queryutil/

BODY raw

```
{
    "QUERY": "
    SELECT
        A1_COD,
        A1_NOME AS NOME
        FROM SA1990
    "
}
```

Example Request Query

```
curl --location --request POST 'http://localhost:8022/apipdv/queryutil/' \
--data-raw '{
    "QUERY": "
    SELECT
        A1_COD,
        A1_NOME AS NOME
        FROM SA1990
    "
```

ANEXO A – Licença MIT