UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Recursos Hídricos

Dissertação

Monitoramento hidrometeorológico frente a ocorrência de desastres naturais: Concepção e implementação de uma API REST para um sistema de gerenciamento de dados

Jamilson do Nascimento

Pelotas, 2025

Jamilson do Nascimento

Monitoramento hidrometeorológico frente a ocorrência de desastres naturais: Concepção e implementação de uma API REST para um sistema de gerenciamento de dados

Dissertação apresentada ao Programa de Pós Graduação em Recursos Hídricos do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Recursos Hídricos.

Orientador: Gilberto Loguercio Collares

Coorientador: Mateus Madail Santin

Pelotas, 2025

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação da Publicação

N244m Nascimento, Jamilson do

Monitoramento hidrometeorológico frente a ocorrência de desastres naturais [recurso eletrônico] : concepção e implementação de uma API REST para um sistema de gerenciamento de dados / Jamilson do Nascimento ; Gilberto Loguercio Collares, orientador ; Mateus Madail Santin, coorientador. — Pelotas, 2025.
76 f.

Dissertação (Mestrado) — Programa de Pós-Graduação em Recursos Hídricos, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2025.

1. API REST. 2. Desastres Hidrológicos. 3. Django REST framework. 4. Redes de monitoramento. I. Collares, Gilberto Loguercio, orient. II. Santin, Mateus Madail, coorient. III. Título.

CDD 627

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Jamilson do Nascimento

Monitoramento hidrometeorológico frente a ocorrência de desastres naturais: Concepção e implementação de uma API REST para um sistema de gerenciamento de dados

Dissertação aprovada, como requisito parcial para a obtenção do grau de Mestre em Recursos Hídricos, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da defesa: 06 de março de 2025

Banca examinadora:

Prof. Dr. Gilberto Loguercio Collares (orientador)
Doutor em Ciência do Solo pela Universidade Federal de Santa Maria – UFSM

Dr. Mateus Madail Santin (coorientador)
Doutor em Educação em Ciência Química da Vida e Saúde pela Universidade Federal do Rio Grande do Sul – UFRGS

Prof. Dr. Felipe de Souza Marques
Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul – UFRGS

Drª. Luciana Shigihara Lima
Doutora em Sensoriamento Remoto pelo Instituto Nacional de Pesquisas Espaciais – INPE

Dedico, a minha mãe, Maria dos Santos Nascimento, meu pai Antônio Eustaquio do Nascimento (*in memoriam*), minhas irmãs, Joana D. do Nascimento, Janaina do Nascimento, Jacqueline do Nascimento, Josilaine do Nascimento.

Agradecimentos

Ao meu orientador e coorientador, *Prof. Dr. Gilberto Loguercio Collares* e *Dr. Mateus Madail Santin,* pela confiança e ensinamentos compartilhados ao longo desses anos.

Aos meus colegas de trabalho, *Rafael F. U. Ehlaert*, *Dr. Nelva Bugoni Riquetti, Arlene Fehrenbach, Tiago Duarte, Patricia Oliveira e Nádia Leal*, pela amizade e conversas em momentos de descontração.

Ao Prof. Dr. George M. S. Gonçalves, por ter me visto, acompanhado e aconselhado, jamais esquecerei o quanto suas ações me ajudaram.

Aos meus amigos de longa data, *Murilo Freitas Bauth, Estephan Araujo Costa* e *Lucas Aurélio de Carvalho.* Obrigado por me provarem a existência de amizades verdadeiras, apesar da distância e correria da "vida adulta".

Aos meus amigos, *Hugo Venâncio Lopes* e *Amanda Cristina Mariano,* pessoas que Pelotas e a UFPEL me proporcionaram a alegria e privilégio de convivência diária durante minha graduação.

À *Victoria Silveira Jonko, Têmis Ester Garcia Corrêa* por me proporcionarem uma nova perspectiva, me fazendo lembrar que existe vida além das obrigações acadêmicas e que é preciso e saudável conciliar obrigações e lazer.

A *César Cardoso Matias,* que me fez lembrar a importância de uma simples pergunta: "*tudo bem ?*". A *Edmar Araujo,* por me colocar de volta do caminho que acabei me desviando involuntariamente e sem perceber, que com toda certeza, não conseguiria voltar sozinho.

As pessoas a quem dediquei este trabalho. Por apoiarem minhas decisões e acreditarem em mim. Sempre presentes, apesar da distância, em todos momentos que necessitei.

À *Universidade Federal de Pelotas, à Agência de desenvolvimento da Bacia da Lagoa Mirim,* ao grupo de pesquisas *NEPE-HidroSedi*, ao *Programa de Pós-Graduação em Recursos Hídricos* e à *Coordenação de Aperfeiçoamento Pessoal de Nível Superior (CAPES)*.

“O inferno são os outros.”

Jean-Paul Sartre

“Das coisas na as coisas.”

Bruno Murani

*“...bust a few rhymes so **motherfuckers** remember where the thought is, I brought all this, so you can survive when law is lawless. Feelings, sensations that you thought was dead. No squealing and remember **that it’s all in your head**…”*

Gorillaz – Clint Eastwood

Resumo

Nascimento, Jamilson. **Monitoramento hidrometeorológico frente a ocorrência de desastres naturais**: **Concepção e implementação de uma API REST para um sistema de gerenciamento de dados.** 2025. Nf 73. Dissertação (Mestrado em Recursos Hídricos) - Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2025.

Diante os desafios ocasionados pelas mudanças climáticas, o monitoramento hidrometeorológico em tempo real apresenta um papel crucial para o combate e mitigação dos efeitos de desastres naturais de cunho hidrológico. No Brasil, a Agência Nacional das Águas e Saneamento Básico (ANA), desempenha a função de monitorar pontos de interesse da União, através do uso de Plataformas de Coletas (PCDs). As PCDs devem ser responsáveis pelo monitoramento continuo de variáveis hidrológicas e elementos climatológicos, tais dados devem ser transmitidos a uma central que é responsável pelo processamento, organização e divulgação das informações obtidas através dos dados monitorados. Em pequenos municípios, é comum que o monitoramento ocorra por organizações governamentais, setores privados e instituições federais de ensino superior, em grande parte dos casos, a coleta dos dados hidrometeorológicos é realizada de maneira manual, necessitando da presença de técnicos especializados. As tecnologias *General Packet Radio Service* (GPRS), o protocolo de comunicação *HyperText Transfer Protocol* (HTTP) e a implementação *Application Programming Interface* (API) baseados na arquitetura *Representational State Transfer* (REST) – API REST têm se mostrado eficientes na concepção de Sistemas Web (SW) para a transmissão de dados em tempo real. Desta forma, o presente trabalho tem o intuito de explicitar o processo de desenvolvimento de uma API REST, através da modelagem orientada a domínio com o uso *dos frameworks Django* e *Django REST framework* (DRF). No primeiro, é levantado a introdução ao contexto do trabalho, seguindo pela exposição dos objetivos. Foram apresentados os conceitos de Arquitetura Sistemas (AS) e Design de Sistemas, no qual abordam diferentes perspectivas em escala do funcionamento de sistema computacional (SW). Após conceitos gerais de AS e DS, são apresentados conceitos fundamentais da arquitetura REST, onde o principal objetivo é o retorno de dados estruturados em formato JSON, e o funcionamento de DRF. O estudo tem como resultado a explicitação o processo de implementação da API REST para um sistema de gerenciamento de dados hidrometeorológicos, através da metodologia modelagem orientada a domínio e os testes de desempenho, com diferentes cenários de quantidade e estruturação dos dados.

Palavras-Chave: API REST, Desastres Hidrológicos, *Django REST framework* , Redes de monitoramento.

Abstract

Nascimento, Jamilson. **Hydrometeorological Monitoring in the Face of Natural Disasters: Conception and Implementation of a REST API for a Data Management System.** 2025. NF 73. Dissertation (Master's in Water Resources) – Center for Technological Development, Federal University of Pelotas, Pelotas, 2025**.**

Given the challenges posed by climate change, real-time hydrometeorological monitoring plays a crucial role in combating and mitigating the effects of hydrological natural disasters. In Brazil, the *Agencia Nacional das Águas e Saneamento Básico* (ANA) is responsible for monitoring points of interest of the Union through the use of *Plataforma de Coleta de Dados* (PCDs). These PCDs are tasked with the continuous monitoring of hydrological variables and climatological elements; the collected data must be transmitted to a central unit responsible for processing, organizing, and disseminating the information obtained from the monitored data. In small municipalities, it is common for monitoring to be carried out by other government organizations, private sectors, and federal higher education institutions; however, in many cases the data collection is performed manually, requiring the presence of specialized technicians.

Technologies such as General Packet Radio Service (GPRS), the HyperText Transfer Protocol (HTTP), and the implementation of an Application Programming Interface (API) based on the Representational State Transfer (REST) architecture – REST API – have proven efficient in the design of Web Systems for real-time data transmission. Thus, the present work aims to elucidate the process of developing a REST API through domain-driven modeling using the Django and Django REST frameworks. The first section introduces the context of the work, followed by an exposition of the objectives. The concepts of Systems Architecture (SA) and Systems Design (SD) are presented, addressing different perspectives on the scale at which a computer system operates. After presenting the general concepts of SA and SD, fundamental concepts of the REST architecture are introduced, with the primary objective of returning data structured in JSON format, along with an explanation of how the Django REST framework (DRF) functions.

Next, the domain-driven modeling methodology is presented, outlining its main concepts. Within the domain context, topics such as the hydrological cycle, watershed, hydrology, hydrological monitoring networks, and the management of hydrological disasters are discussed. The study culminates in elucidating the process of implementing the REST API for a hydrometeorological data management system, employing domain-driven modeling and performance tests under different scenarios regarding data quantity and structure.

Keywords: REST API, Hydrological Disasters, Django REST framework, Monitoring Networks.

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de abreviaturas e siglas

ANA	Agencia Nacional das Águas e Saneamento Básico
API	Application Programming Interface
AS	Arquitetura de Sistemas
CEMADEN	Centro de Nacional de Monitoramento e Alertas de Desastres Naturais
DDD	Domain Driven Design
DRY	Don't Repeat Yourself
DS	Design de Sistemas
GPRS	General Packet Radio Service
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MVC	Model - View - Controller
MVT	Model - View - Template
ORM	Object Relational Mapping
PCDs	Plataforma Coleta de Dados
PNDEC	Política Nacional De Proteção e Defesa Civil
PNRH	Política Nacional de Recursos Hídricos
PO	Ponto de Observação
POO	Programação Orientada a Objetos
RBS	Revisão Bibliográfica Sistemática
REST	Representational State Transfer
SC	Sistema Computacional
SEMA	Secretaria do Meio Ambiente e Infraestrutura
SGD	Sistema de Gerenciamento de Dados
SINGREH	Sistema Nacional de Gerenciamento de Recursos Hídricos
SQL	Structured Query Language
SW	Sistema Web
TCP/IP	Transmission Control Protocol/Internet Protocol

Sumário

1 Introdução

O monitoramento hidrometeorológico tem entre seus objetivos, a obtenção de um conjunto de dados registrados ao longo do tempo, que possibilitem a caracterização do comportamento de determinada bacia hidrográfica, frente a ocorrência de fenômenos presentes no ciclo hidrológico. Tal atividade desempenha papel central no enfrentamento de desastres hidrológicos, como secas e inundações.

No caso secas, o monitoramento contínuo auxilia na gestão assertiva dos recursos hídricos, com objetivos de evitar colapso no abastecimento, fornecendo informações cruciais para o planejamento do consumo de água em seus diversos usos, como atividades agrícolas, indústrias, recreativas e consumo humano. Em casos de enchentes, alagamentos e inundações, o monitoramento contribui para a elaboração de planos de ação e enfretamento, possibilitando evacuações em áreas de risco e a implementação de medidas preventivas visando a minimização de danos e riscos a vida da população.

Os dados registrados por meio de monitoramento podem ser realizados de maneira automática, por meio de sensores, sendo armazenados em dispositivos de memória e transmitidos para um sistema central, através do uso das plataformas automáticas de coleta de dados (PCDs). Segundo a definição da Agência Nacional das Águas e Saneamento Básico (ANA), uma PCDs deve ser composta por: sensor de chuva, sensor de nível d'água, sensores de elementos climatológicos, sistema de alimentação por captação de energia solar, regulador de carga de bateria, sistema de transmissão de dados por satélite ou *General Packet Radio Service* (GPRS), e *datalogger*. O sistema de transmissão deve se comunicar com um sistema central, onde seus dados são armazenados, processados e posteriormente utilizados na produção de informações como subsídio para os tomadores de decisão (ANA, 2011).

O protocolo de comunicação *Hypertext Transfer Protocol* (HTTP) em conjunto com a implementação uma *Application Programming Interface* (API) com arquitetura *Representational State Transfer* (REST) – API REST, tem se mostrado eficiente em sistemas de transmissão de dados estruturados em formato JSON por meio de tecnologia GPRS.

Thaines (2022) relata sucesso na implementação de um sistema de transmissão com tal abordagem, em uma aplicação mobile de auxílio na obtenção de valores de evapotranspiração de referência para manejo de irrigação agrícola. Assim também, Zeng, Tu e Ma (2011) evidenciam a eficiência do sistema na obtenção de dados para o controle de enchentes no lago Poyang na província de Jiangxi, na China.

Kermmerich (2019) refere a sua experiência como satisfatória, no desenvolvimento de um sistema de telemetria via GPRS e arquitetura REST, para uma estação de monitoramento hidrometeorológico, com registros de temperatura, umidade relativa e nível d'água em um reservatório artificial experimental, realizados em intervalo de tempo de 5 minutos.

Moreira (2024), descreve o processo de desenvolvimento de um aplicativo *web* para a visualização de dados hidrológicos, implementando uma interface de interação com o usuário, e uma API REST para armazenamento, aquisição e apresentação de dados hidrológicos.

O passo inicial no desenvolvimento de sistemas em suas diversas abordagens, consiste na identificação de seu domínio. O domínio de um sistema consiste em um conjunto de problemas, soluções, regras e elementos atuantes no contexto geral de funcionamento do sistema. Determinado o domínio, estabelece-se a arquitetura e metodologia de design da implementação. A arquitetura de Sistemas (AS) abrange uma perspectiva geral do funcionamento, enquanto o Design de Sistemas (DS) abrange aspectos de codificação e padrões de interação. Neste processo, é fundamental a interação de um agente interlocutor especialista no contexto do domínio, e o agente responsável pelo desenvolvimento do sistema (Evans, 2004; Martin; Grennin; Brown, 2019; Percival; Gregory, 2020).

Diante ao exposto, o presente trabalho tem como objetivo elucidar e o processo de desenvolvimento de uma API REST através da metodologia de *design,* Modelagem orientada a domínio, para um sistema de transmissão de dados entre uma PCDs e seu sistema central, visando otimizar a coleta, transmissão e produção de informações, que auxiliem em estudos frente a ocorrência de desastres hidrológicos, como secas e inundações, e demais áreas correlacionadas a gestão de recursos hídricos.

1.1 Objetivos

1.2 Geral

Implementar uma API REST para uma rede de monitoramento hidrometeorológico, através da metodologia *design* de sistemas Modelagem orientada a domínio. Considerando a hidrologia, a bacia hidrográfica e o monitoramento hidrometeorológico e seu papel no combate de desastres naturais como contexto do domínio para um sistema de gerenciamento de dados hidrometeorológicos (SGD).

1.3 Específicos

- Realizar um levantamento teórico e bibliográfico sobre arquitetura e design de sistemas, hidrologia, monitoramento hidrológico e estratégias de combate a desastres hídricos, com o intuito de fundamentar a pesquisa e identificar abordagens relevantes para a temática estudada.
- Analisar e selecionar o material previamente levantado, tendo como critério de eliminação a coerência dos resultados e discussões nos documentos analisados, com os objetivos estabelecidos na presente dissertação.
- Realização do relato da metodologia utilizada na implementação da API REST, explicitando o método de modelagem orientada a domínio.
- Implementação de uma API REST capaz de receber dados em formato .JSON.
- Avaliar a melhor desempenho de acordo com estruturação dos dados .JSON.

2 Revisão da Literatura

2.1 Arquitetura e Design de sistemas

Um Sistema Computacional (SC), consiste em um conjunto organizado de componentes que interagem entre si. O SC recebe um estimulo de entrada do qual é processado, gerando de informações que podem ser armazenadas e/ou transmitidas a outros SC. Entende-se como componentes de um SC o *hardware* (componentes físicos), *software* (componente lógico), dados, usuários e rede de compartilhamento (Stair; Reynalds, 2016; Kalinowski *et al.*, 2023).

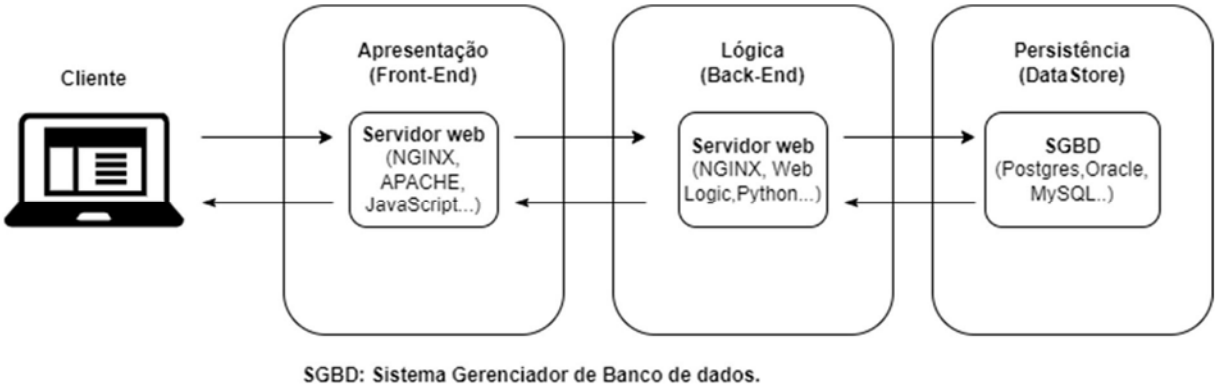
Os conceitos de AS e DS são ambíguos, não existe uma definição concreta na literatura, uma vez que existe a possibilidade de intersecção no produto resultante de ambas as partes. Suas nuances variam de acordo com a época, infraestrutura, tecnologia e a metodologia empregada em sua concepção e implementação. Entretanto, existe um consenso encontrado em diferentes obras de diferentes autores, no qual a AS é tida como a abstração da visão geral do funcionamento do sistema e DS como uma visão mais detalhada de seu funcionamento (Fielding, 2000; Tsuruta, 2010; Percival, Gregory 2020).

De acordo com Pressman e Maxim (2021), a AS é focada nos componentes de *software*. Um componente de *software* possui diferentes níveis de complexidade, onde neste nível suas propriedades internas a nível de algoritmo não são detalhadas. Os autores destacam que diferentes estilos de AS possuem características em comum, sendo, um conjunto de componentes que realizam as demandas do sistema, um conjunto de conectores responsáveis pela comunicação entre os componentes, restrições de interação entre os componentes e modelos semânticos (gráficos). Em SC tradicionais, é comum que que seu funcionamento a AS se baseie inicialmente em três camadas de contexto de atuação, sendo elas: Apresentação, Lógica e Persistência. A camada de Apresentação lida com a interação do usuário com o sistema, a camada de Lógica lida com as regras, validações, permissões e cálculos so sistema, e por fim, a camada de Persistência lida com o acesso e armazenamento dos dados do sistema, seja em banco de dados, arquivos, entre outros. A AS em três camadas (Figura 1) é comumente utilizada em Sistemas Web (SW). Destaca-se que

o sistema é representado com alto nível de abstração (Grennin; Brown, 2019; Moreira, 2024).

.

Figura 1 – AS para um sistema web dividida em três camadas



Fonte: Adaptado Percival e Gregory (2020)

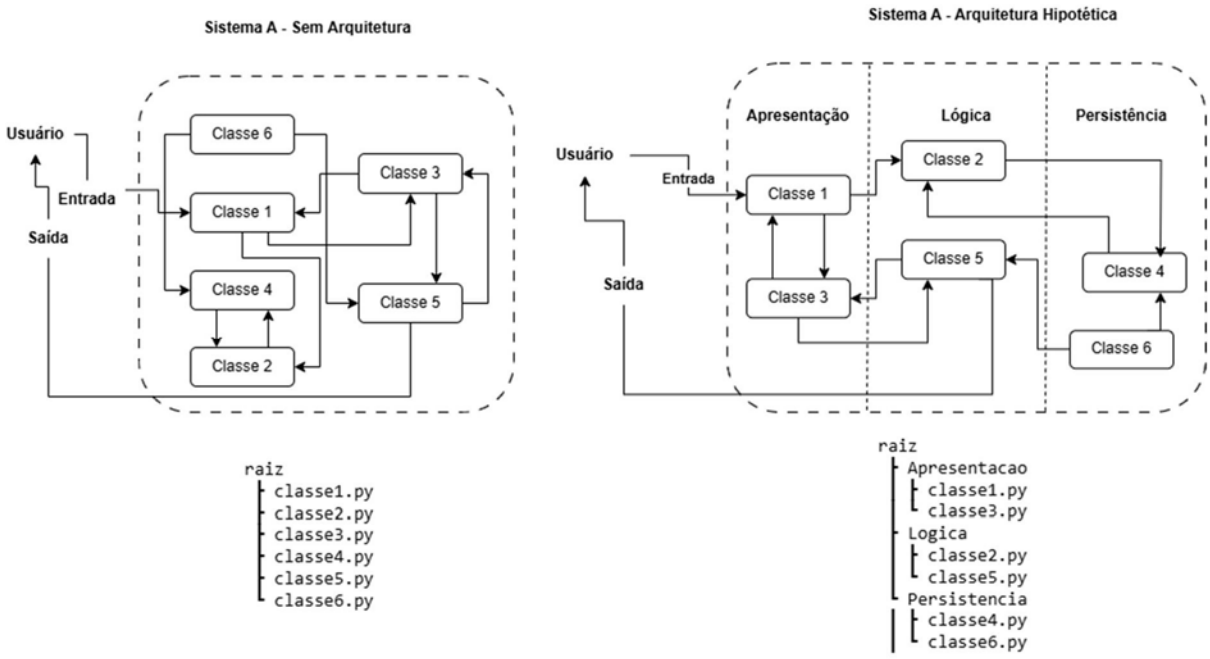
Diversos autores defendem diferentes abordagens para a implementação de um SW considerando AS e DS, onde o nível de detalhamento e metodologias sugeridas dependem dos componentes presentes no SW. Com a popularização e o aumento do acesso à internet no início dos anos 2000, houve um aumento do uso de linguagens que suportam o paradigma de programação orientada a objetos (POO) para a criação de SW. As vantagens de se utilizar tais linguagens, além do grande número de usuários dando suporte a comunidade de desenvolvedores, é a existência de bibliotecas e *frameworks* que fornecem um conjunto de códigos e estruturas pré-definidas que auxiliam o desenvolvimento. Tais como: *Django – Python*, *Ruby on Rails – Ruby*, *Spring – Java*, *Laravel – PHP*, *ASP.NET Core – C#.*

O desenvolvimento de SW considerando POO consiste na implementação de objetos, ligados ao contexto da finalidade do sistema, tal contexto é usualmente chamado de domínio do sistema. Um objeto é definido por meio de classes, que consistem no agrupamento de atributos (dados) e métodos (comportamento), com objetivo de representar determinado elemento da vida real ou elementos abstratos (Percival; Gregory, 2020; Github, 2024).

Quando uma classe é instanciada, origina-se um objeto, que é a representação concreta desta classe, contendo atributos específicos e ocupando espaço em endereço de memória único. Desta forma, um sistema pode ser implementado por um conjunto de classes, que interagem entre si, recebendo informações de entrada, processando-as, gerando informações de saída (Percival; Gregory, 2020; Lott; Philips, 2021). Um exemplo de um conjunto de Classes para um SW, considerando a ausência

camadas de contexto e um SW com camadas de contexto organizados em uma arquitetura hipotética (Figura 2).

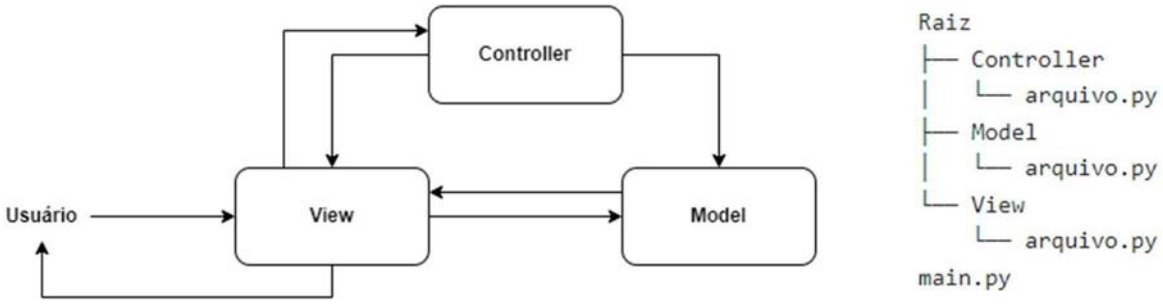
Figura 2 – Exemplo de disposição das classes em um sistema com design hipotético.



Fonte: Adaptado Percival e Gregory (2020)

SWUm dos padrões de AS mais populares no desenvolvimento de *software* e a arquitetura *Model-View-Controller* (MVC). Foi apresentada em 1970, por Trygve Reenskaug, com foco na interação do usuário com o sistema (Figura 3). A camada *Model* representa os dados e a lógica do sistema, a camada *View* é responsável pela apresentação de dados ao usuário, e a camada *Controller* é responsável pelo gerenciamento da interação do sistema com o usuário, processando informações de entrada entre as camadas *Model* e *View* (Rocha, 2018; Mantovani, 2021; Reenskug, 2024).

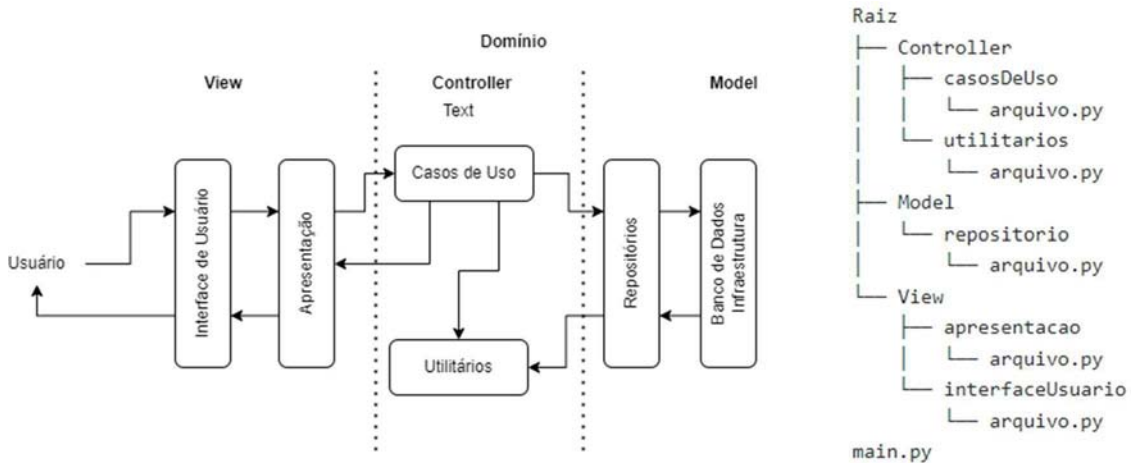
Figura 3 – Arquitetura MVC



Fonte: Adaptado (Rocha 2018; Mantovani, 2021)

A Figura 3 ilustra um SW a nível de *software* de modo geral, considerando três camadas da arquitetura MVC e a comunicação realizadas entre si. A Figura 4 apresenta a mesma arquitetura, porém considerando DS empregado na implementação. Desta forma, é representado entidades comuns empregas com a técnica de DS “Modelagem orientada a domínio”.

Figura 4 – Arquitetura MVC idealizada com classes comum no design considerando a Modelagem orientada a domínio.



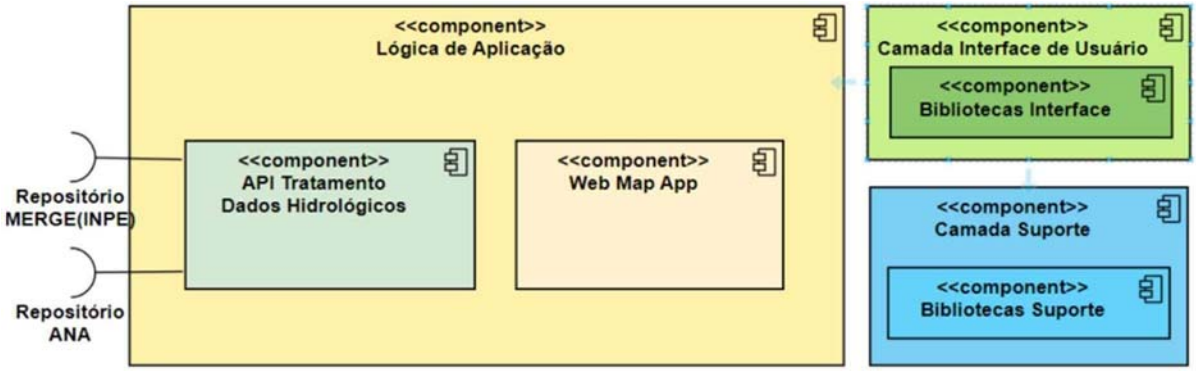
Fonte: Adaptado (Rocha, 2018; Percival; Gregory, 2020; Mantovani, 2021)

Falando de forma resumida, DS em SW a nível de *software* são metodologia carregada de subjetividade do desenvolvedor e da metodologia empregada durante os processos de concepção e implementação. Todos autores (sem exceção) citados nesta seção, em algum momento defendem ou relatam a adaptabilidade de qualquer técnica empregada, ressaltando que não existe uma “lei” de empregabilidade vigente. Desta forma, a técnicas de AS e DS assumem um papel de fornecer uma orientação entre a comunicação entre a comunidade de desenvolvedores, estabelecendo

diretrizes já testadas e consolidadas que melhor de adequem ao SW a ser desenvolvido. .

Moreira (2024) implementa um SW para a visualização de dados hidrológicos (Figura 5), considerando o *back-end* e *front-end.* Em sua metodologia adotada para AS, o SW é representado em alto nível de abstração. O autor minimiza as responsabilidades na camada de lógica no *back-end*, desta forma utilizando as estruturas padrões do *framework,* sem a aplicação de DS (Figura 6). Uma vez que o intuito principal da aplicação se concentra na visualização dos dados disponibilizados por *APIs* de terceiros. O papel que seria da DS é descrito em seu trabalho como técnicas de experiência do usuário, com foco na interação usuário-sistema.

Figura 5 – Arquitetura de Alto nível do sistema de visualização de dados hidrológicos



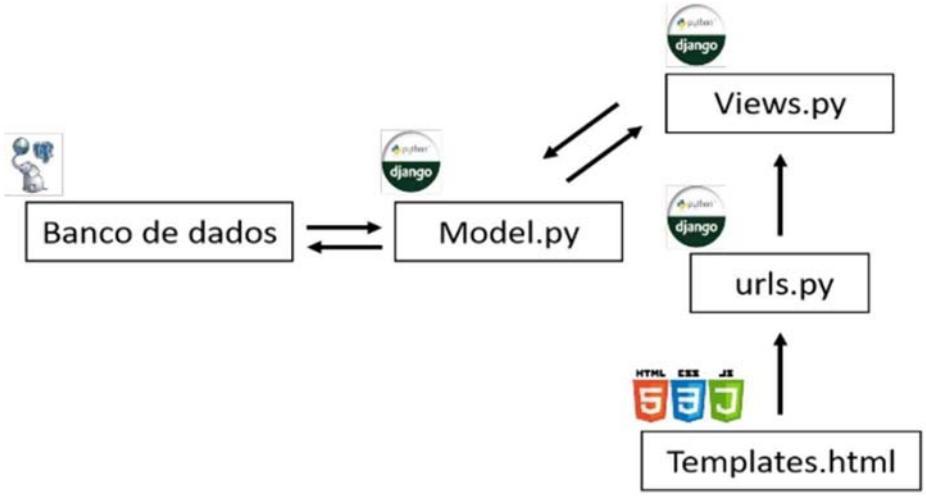
Fonte: Moreira (2024)

Figura 6 – Arquitetura de alto nível com a especificação das tecnologias envolvidas no desenvolvimento do SW.

Fonte: Moreira (2024)

No trabalho efetuado por Lima, Freiman e Camboim (2022), os autores apresentam um SW de mapeamento colaborativo para áreas de risco em alagamento no munício de nova Friburgo – RJ. Os autores utilizam técnicas de AS utilizando a arquitetura MVT (*Model-View-Template*) (Figura 7), sem apresentar DS, sendo o domínio da aplicação reduzido as possibilidades de ações executadas por diferentes tipos de usuários (**Erro! Fonte de referência não encontrada.**). A arquitetura MVT é uma adaptação da arquitetura MVC que disponibiliza os dados diretamente em arquivos *Hypertext Markup Language* (HTML) (Mozilla, 2024).

Figura 7 – Arquitetura em alto nível da SW de mapeamento colaborativo de áreas de risco.



Fonte: *Adaptado (*Lima; Freiman; Camboim, 2022)

A utilização de AS e DS é uma forma de garantir a comunicação e padronização entre os desenvolvedores e identificação de componentes e funcionalidades comuns, apesar a subjetividade aplicada durante o desenvolvimento. O desenvolvimento de SW sem planejamento de uma AS ou DS (técnicas de engenharia de *software*) é conhecida pela comunidade de desenvolvedores como "*Go Horse"* ou "Programação Orientada ao Improviso". Esta forma de desenvolvimento bastante utilizada na manutenção e refatoração de sistemas legados, com tecnologia antiga, que durante a sua manutenção e/ou refatoração os sistemas deve permanecer em produção.

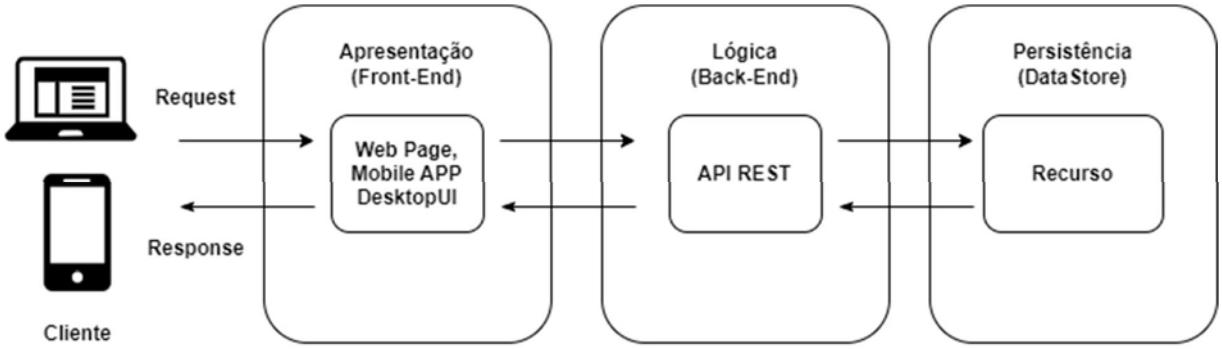
2.1.1 Arquitetura REST e Django

Uma API é definida como um conjunto de protocolos, regras e convenções que auxiliam na comunicação entre diferentes SW. De modo geral, uma API tem o papel de abstrair a complexidade interna de um SW, expondo apenas os métodos, funções e dados necessários para que outros SW possam utilizar suas funcionalidades (Fielding, 2000; Mozilla, 2024).

No contexto de SW, a *API REST* é um conjunto de restrições e convenções de desenvolvimento que tem por objetivo a simplificação da comunicação entre cliente e servidor através dos protocolos *Transmission Control Protocol/Internet Protocol* (TCP/IP) e HTTP. O protocolo TCP/IP forma a base de comunicação da internet, definindo como os dados são empacotados, direcionados e transportados nos sistemas conectados a uma rede, o TCP garante que os dados cheguem de forma correta e completa, e o IP é responsável pela identificação das máquinas para quais os dados devem ser enviados. O protocolo HTTP opera sobre a camada de transporte do TCP/IP, carregando páginas *web* baseados HTML, JavaScript e CSS (Hillar, 2018; Vicent, 2022; Mozilla, 2024).

O princípio de funcionamento de uma API REST se baseia na realização de requisições por parte do cliente e respostas por parte do servidor (*Request/Response*). As requisições utilizam os verbos HTTP padronizados para a realização de ações nos recursos da API REST hospedada no servidor (Figura 8). Define-se como recurso, os dados registrados em banco de dados. Cada recurso pode ser acessado através de um endereço de *url* único, e caracterizado por meio de um número de identificação *id* (Hillar, 2018; Vicent, 2022; Mozilla, 2024).

Figura 8 - Representação do princípio de funcionamento de uma API REST



Fonte: Adaptado Hillar (2018)

O Quadro 1 descreve os principais verbos HTTP utilizados em API REST, sendo responsáveis peta criação, busca/leitura, atualização total, atualização parcial e remoção de recursos do sistema.

Quadro 1 – Verbos/Métodos HTTP comumente utilizados em API REST

Verbo	**Método HTTP**	**Descrição**	**url**
Create	POST	Envia dados ao servidor e criar um novo recurso.	api/v1/recursos
Read	GET	Solicita a representação de um recurso. Não altera o estado do recurso no servidor.	api/v1/recursos
			api/v1/recursos/id
Update	PUT/PATCH	Usados para atualização total ou parcial de um recurso no servidor, respectivamente.	api/v1/recursos/id
Delete	DELETE	Remove um recurso no servidor.	api/v1/recursos/id

Fonte: Adaptado (Hillar, 2018; Vicent, 2022; Mozilla, 2024).

Cada requisição realizada ao servidor contém todas as informações necessárias sem depender de requisições anteriores, uma vez que o servidor não armazena estado de informação entre as requisições. A resposta realizada por servidor retorna determinado recurso em formato *JavaScript Object Notation* (JSON) e um código de *status* (Tabela 1), que informa o resultado da requisição. Os códigos de *status* são divididos em cinco categorias, indicadas pelo primeiro dígito, tendo cincos *status* principais (Tabela 2) que ocorrem com maior frequência .

Tabela 1 – Categorias dos códigos de status das respostas do protocolo de comunicação HTTP.

Código	Categoria	Descrição
1xx	Informativo	Requisição recebida e em processamento.
2xx	Sucesso	Requisição recebida e bem-sucedida.
3xx	Redirecionamento	Requer medidas adicionais do cliente para finalização do processamento.
4xx	Erro Cliente	Erro ao realizar uma requisição feita pelo cliente.
5xx	Erro Servidor	Erro no servidor ao processar a requisição.

Fonte : Mozilla (2024)

Tabela 2 – Códigos de status mais utilizados em APIs REST.

Código	Status	Descrição
200	OK	Solicitação bem-sucedida.
201	Created	Recurso criado com sucesso.
400	Bad Request	Requisição inválida.
404	Not Found	Recurso não encontrado.
500	Internal Server Error	Erro interno no servidor.

Fonte: Adaptado (Hillar, 2018; Vicent, 2022; Mozilla, 2024)

O formato JSON consiste em um formato de dados de em texto, que permite a troca de informações entre diferentes sistemas. É uma das principais formas de troca de informações utilizados na *web,* se trata de uma estrutura organizada em pares chave-valor, da qual a chave é obrigatoriamente sempre *string* definida em as duplas. O valor permite dados do tipo: *String, Int, Boolean, Null, Object e Array* (Mozilla, 2024).

Figura 9 – Exemplo de estrutura JSON com todos os tipos de dados permitidos

Fonte: Autor.

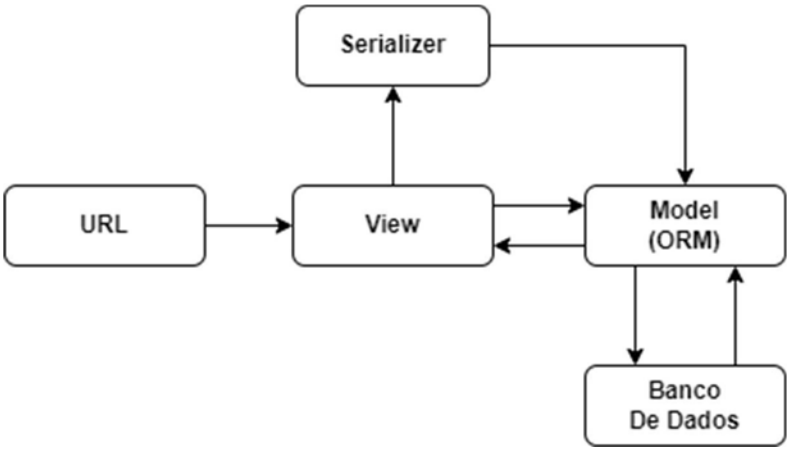
Segundo Hillar (2018), durante o processo de concepção de uma API REST, o primeiro passo é se ter em mente que nem todo usuário deve ter acesso pleno aos recursos da API, que se deve ter um sistema de autenticação e permissões. Vicent (2022) apresenta três alternativas para a implementação, *OAuth*, *JWT* e *APIKeys.* Cada uma necessita de suas peculiaridades no momento de implementação, oferecendo vantagens e desvantagens. O autor salienta, que ao se utilizar ferramentas terceiras, como os diversos *frameworks web* disponíveis, a utilização de ferramentas nativas presentes quase sempre é a melhor opção para a implementação.

Django é um *framework web* Python, de alto nível de abstração que tem por objetivo o desenvolvimento de aplicações *web.* Foi criado em 2005 por Adrian Holovaty e Simon Willison, seguindo as diretrizes de desenvolvimento DRY – *Don't Repeat Yourself,* que incentiva a criação de componentes reutilizáveis em um SW. *Django é* popular entre desenvolvedores web devido as suas principais funcionalidades nativas: sistema de autenticação, *Object Relacional Mapping* (ORM)

que permite a manipulação dos dados registrados no banco com através do Python, painel de administração e suporte a diversos sistemas gerenciadores de bancos de dados (Melé, 2024; Mozilla 2024).

O *Django* opera sobre arquitetura MVT, similar ao padrão MVC, a responsabilidade de controle das interações do usuário é atribuída a *View* e os dados são apresentados diretamente ao usuário através de *Template HMTL*. *Django REST framework* (DRF) desmonta essa arquitetura (Figura 10), onde os dados (agora tratados como recursos) não são mais apresentados ao usuário diretamente no *Template HTML* e sim a camada de *front-end* através das *urls* (Melé, 2024; Mozilla 2024).

Figura 10 - Arquitetura funcionamento DRF.



Fonte: Mozilla (2024)

É adotado pela comunidade uma diretriz de implementação de SW quando utilizado DRF (Quadro 2). Os passos adotados seguem o que se convencionou como implementação inicial mais amigável para iniciantes, sem levar em consideração qualquer AS diferentes do padrão DRF (Figura 10), obtendo assim a estrutura padrão (Figura 11) de um projeto DRF.

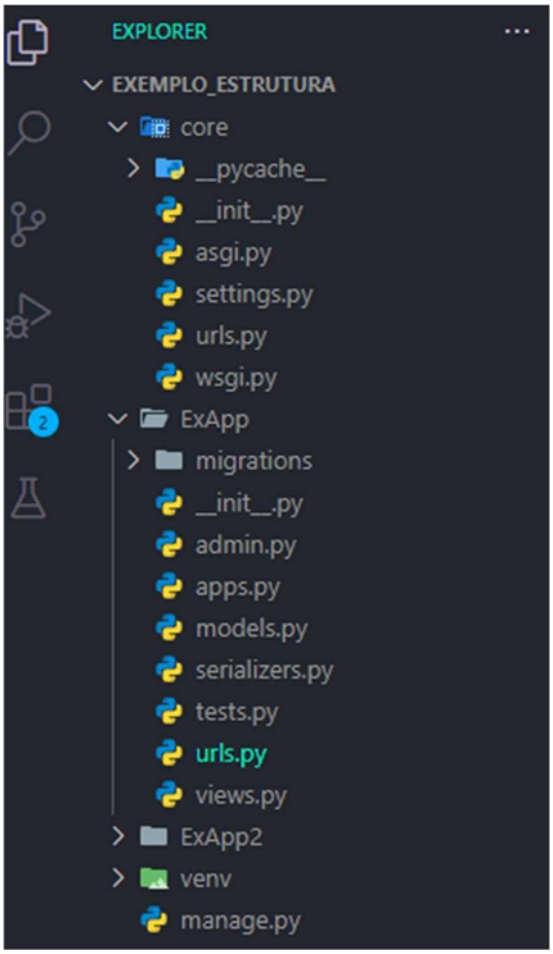
.

Quadro 2 – Diretriz de implementação de uma API REST adotada pela comunidade DRF.

Etapa	Ação	Descrição
1	Configuração inicial.	Criação ambiente virtual de desenvolvimento.
	Iniciar projeto Django.	Instalar Django e DRF.
2	Definição dos modelos.	Diagrama de entidades de Relacionamento.
	Inicializar aplicações DRF.	Aplicação de usuários e demais.
	Implementar modelos.	Codificar modelos em suas respectivas aplicações.
3	Definição dos Serializers.	Codificar serializers de seus respectivos modelos.
4	Definição das Views e urls	Definir qual tipo de view e configurar urls.
5	Configuração autenticação e permissões.	Definir método a ser implementado.
6	Escolha e configuração ferramentas úteis.	Paginação, filtros, cache, testes, monitoramento de performasse, estilo de código, entre outros.

Fonte: Adaptado Mozilla (2024)

Figura 11 – Estrutura padrão ao se inicializar um projeto DRF



Fonte: Autor

Em adição as funcionalidades nativas oferecidas por *Django,* DRF disponibiliza as seguintes funcionalidades nativas que facilitam a implementação de uma API REST: Serializers, responsáveis or transformar objetos Python em formato JSON e vice-versa. *FunctionViews, APIViews, GenericViews e ViewSets* que processam as requisições HTTP. *Routers,* responsáveis pela configuração automática das urls ao se optar pela utilização de *ViewSets.*

Hillar (2018), destaca que o uso de *ViewSets* e *Routers* devem ser utilizadas com cautela, uma vez tais funcionalidades limitam e dificultam a personalização da API, sendo necessário sobrescrever seus códigos, o que elimina as vantagens de utilizá-las. O autor recomenda a utilização quando a API não possui lógica além das operações CRUD. Vicent (2022), indica a utilização de *FunctionViews ou APIViews* para APIs que precisem de um alto nível de customização, ou apresentem lógicas complexas a serem aplicas em seus recursos.

Com a implementação de um modelo genérico no *Django* (Figura 12), sendo a principal implantação, a partir de modelo as demais funcionalidades são implementadas.

Figura 12 – Exemplo de implementação modelo genérico *Django* - DRF

core > apps > ModelApi > models.py > ...

```python
from django.db import models


class ModeloGenericoA(models.models):
    """
    Modelo Generico para exemplificação de método de Implementação
    """
    nome = models.CharField(max_length=100)
    variavel_A = models.IntegerField()
    variavel_B = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return self.nome

```

Fonte: Autor

A implementação de um *serializer* do DRF. O *serializers* são os responsáveis pela transformação de um objeto Python em um objeto com estrutura *JSON* e vice-versa (Figura 13), tais processos são denominados serialização e de-serialização.

Figura 13 – Exemplo implementação *serializer* genérico *Django* - DRF

core > apps > ModelApi > serializers > serializers.py > ModeloGenericoASerializer

```python
from models import ModeloGenericoA
from rest_framework.serializers import ModelSerializer


class ModeloGenericoASerializer(ModelSerializer):
    """
    Serializer Generico para exemplificação de método de Implementação
    """

    class Meta:
        model = ModeloGenericoA
        fiels = "__all__"

```

Fonte: Autor

O principal método de criação de um instancia de determinado modelo se dá através da implementação de uma *APIView* responsável pela execução do método HTTP *CREATE* (Figura 14).

Figura 14 – Exemplo implementação *Views CREATE Django* - DRF

core > apps > ModelApi > views.py > ...

```python
from django.shortcuts import get_object_or_404
from models import ModeloGenericoA
from rest_framework import status
from rest_framework.response import Response
from rest_framework.views import APIView
from serializers.serializers import ModeloGenericoASerializer


# Criar ModeloGenericoA
class ModeloGenericoACreateAPIView(APIView):
    def post(self, request):
        serializer = ModeloGenericoASerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Fonte: Autor

A busca e leitura de modelos consistidos no banco de dados, se através da implementação de duas *APIView* responsáveis pela execução do método HTTP *READ*. Nota-se que a implementação foi realizada visando a busca de uma lista de objetos e um objeto especifico, respectivamente (Figura 15).

Figura 15 - Exemplo implementação *Views READ Django* - DRF

```python
# Listar ModeloGenericoA
class ModeloGenericoAListAPIView(APIView):
    def get(self, request):
        produtos = ModeloGenericoA.objects.all()
        serializer = ModeloGenericoASerializer(produtos, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)


# Buscar ModeloGenericoA Específico
class ModeloGenericoARetrieveAPIView(APIView):
    def get(self, request, pk):
        produto = get_object_or_404(ModeloGenericoA, pk=pk)
        serializer = ModeloGenericoASerializer(produto)
        return Response(serializer.data, status=status.HTTP_200_OK)

```

Fonte: Autor

A atualização de características do modelo ou sua remoção completa do sistema, se dá pela implementação de duas *APIView* (Figura 16) responsáveis pela execução dos métodos HTTP *UPDATE* e *DELETE*, respectivamente.

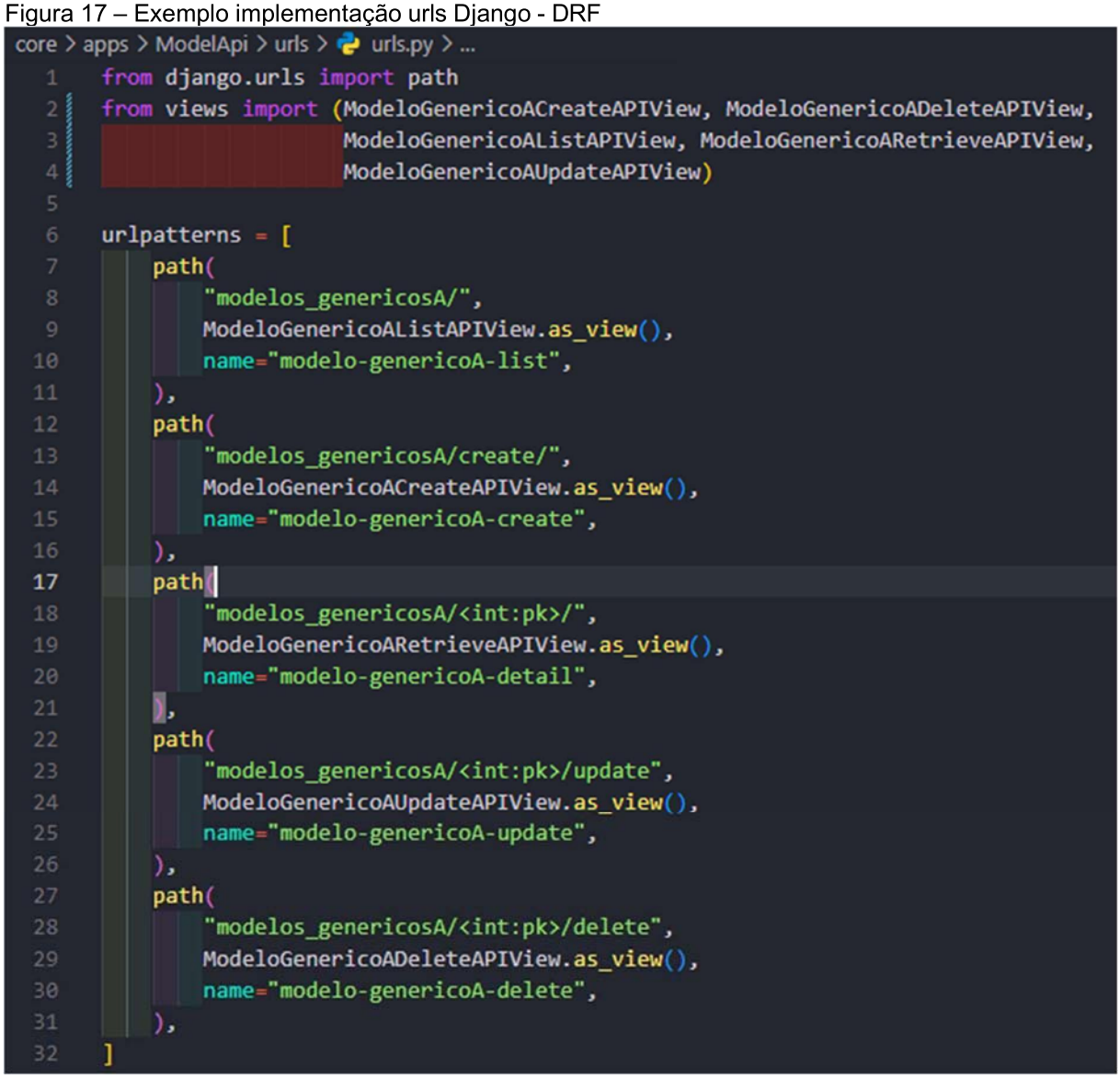
Figura 16 - Exemplo implementação *Views* UPDATE e DELETE *Django* - DRF

```python

# Atualizar ModeloGenericoA
class ModeloGenericoAUpdateAPIView(APIView):
    def put(self, request, pk):
        produto = get_object_or_404(ModeloGenericoA, pk=pk)
        serializer = ModeloGenericoASerializer(produto, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)


# Deletar ModeloGenericoA
class ModeloGenericoADeleteAPIView(APIView):
    def delete(self, request, pk):
        produto = get_object_or_404(ModeloGenericoA, pk=pk)
        produto.delete()
        return Response(
            {"message": "ModeloGenericoA deletado com sucesso"},
            status=status.HTTP_204_NO_CONTENT,
        )
```

Fonte: Autor

A implementação das *urls* (Figura 17), que se utilizam das *Views* para garantir o acesso e manipulação a determinados modelos (recursos) da API REST.

Figura 17 – Exemplo implementação urls Django - DRF



Fonte: Autor

As configurações básicas de Autenticação, permissões, filtragem de dados e paginação a serem configuradas em *settings.py* (Figura 18), seguindo as normas recomendadas da documentação oficial.

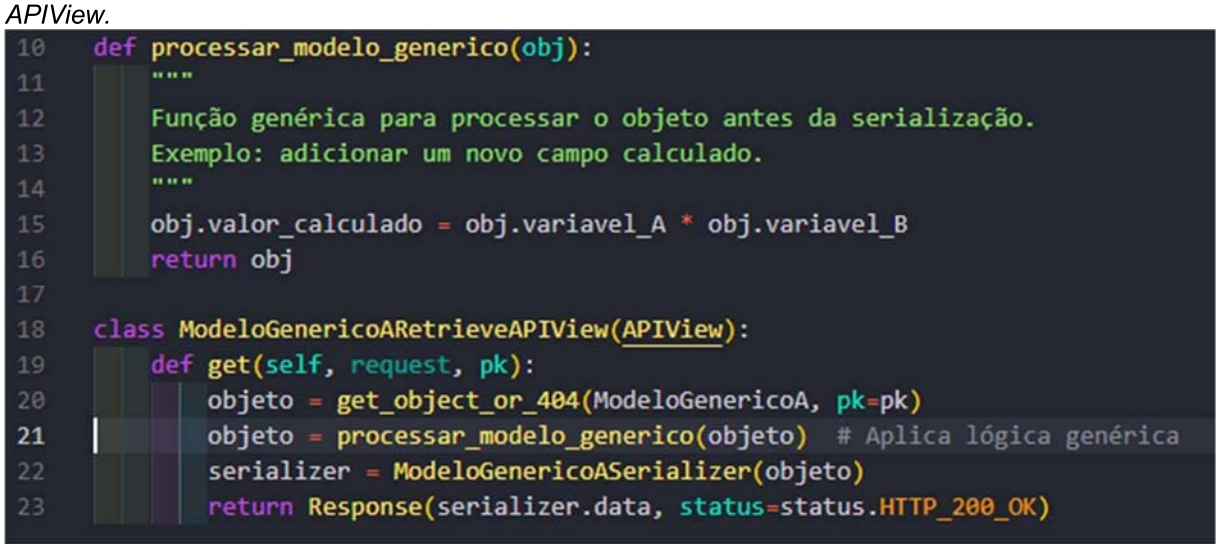
Figura 18 – Configuração Autenticação, permissões filtros e paginação em *settings.py*

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ]
}
REST_FRAMEWORK['DEFAULT_FILTER_BACKENDS'] = ['django_filters.rest_framework.DjangoFilterBackend']
REST_FRAMEWORK['DEFAULT_PAGINATION_CLASS'] = 'rest_framework.pagination.PageNumberPagination'
REST_FRAMEWORK['PAGE_SIZE'] = 10
```

Fonte: Autor

A aplicação de lógica sobre o(s) objeto(s) retornado(s), pode ser realizada com codificação direta no escopo *APIView* (o que não é recomendado) ou através da aplicação de funções distintas (Figura 19).

Figura 19 – Exemplo de implementação de aplicação de lógica sobre o objeto retornado em uma *APIView.*



```
def processar_modelo_generico(obj):
    """
    Função genérica para processar o objeto antes da serialização.
    Exemplo: adicionar um novo campo calculado.
    """
    obj.valor_calculado = obj.variavel_A * obj.variavel_B
    return obj

class ModeloGenericoARetrieveAPIView(APIView):
    def get(self, request, pk):
        objeto = get_object_or_404(ModeloGenericoA, pk=pk)
        objeto = processar_modelo_generico(objeto)  # Aplica lógica genérica
        serializer = ModeloGenericoASerializer(objeto)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

Fonte: Autor

Toda implementação foi realizada de maneira genérica, com objetivo de ilustrar o padrão de desenvolvimento *Django e DRF.* Existem inúmeras maneiras de implementação das *Views* e lógica de aplicação. Cabe ao desenvolvedor descobrir qual confere maior confiabilidade durante o desenvolvimento. O principal desafio é a customização para a implementação de logica e regras.

2.2 Modelagem orientada a domínio

Segundo Brock e McKean (2002), o DS tem origem na década 70, com tentativas de sistematizar a comunicação com outros desenvolvedores. A partir dos da década 90, com a popularização do POO, conceitos de *design patterns* a nível de código de *software* foram apresentados. *Design patterns* são tentativas de padronização de código, para soluções reutilizáveis que se encaixem em diferentes SW. Os autores relatam que a partir deste contexto, a implementação de objetos se torna usual para a representação do domínio de um SW. O principal conceito proposto é que cada objeto do domínio deve ser organizado e atribuído uma responsabilidade colaborativa com os demais objetos.

Gamma *et al.* (1994) definem 23 *designs patterns*, divididos em três categorias, sendo elas, padrões de criação, so quais são responsáveis na criação de objetos. Padrões de estrutura, que lidam com a organização geral dos objetos do SW. E por último, padrões de comportamento, que lida como os objetos interagem entre si. Percival e Gregory (2020), ressaltam que os padrões supracitados foram de grande importância para a área de desenvolvimento de SW baseado em POO. Porém, o autor destaca que grande parte está em desuso, ou utilizados em sistemas legado, devido a evolução de ferramentas que auxiliam e facilitam o desenvolvimento.

A Modelagem orientada a domínio é um conjunto técnica de DS que tem por objetivo a definição do domínio de um SW. Possui um conjunto de técnicas e convenções, com a finalidade de facilitar a comunicação entre o especialista do domínio e desenvolvedores. O especialista do domínio é o agente que possui conhecimento aprofundado em seu contexto, sendo responsável por apontar possíveis entidades e seus comportamentos, regras e fronteiras nas camadas de contexto e entidades (Quadro 3) (Percival; Gregory, 2020).

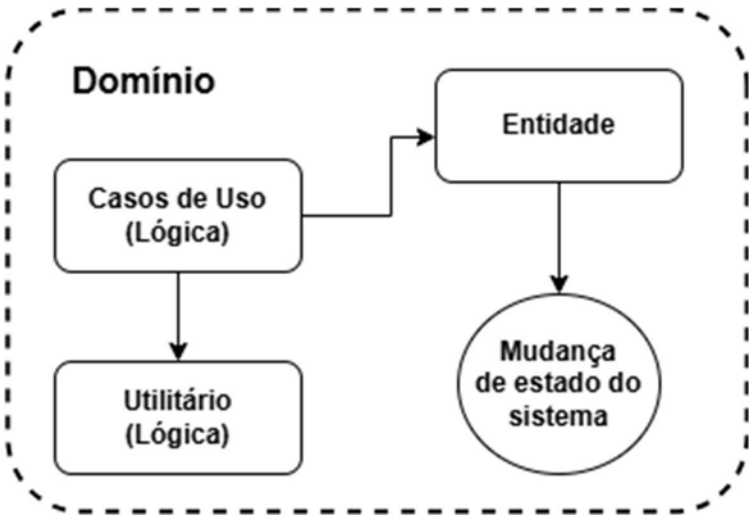
Quadro 3 – Relação de vocabulário inicial para a definição de linguagem ubíqua

Elemento de Domínio	Descrição
Entidade	Qualquer elemento presente no domínio.
Recurso (Resources)	Objetos registrado em banco de dados. Possuem atributos
Caso de Uso (Use Case)	Lógica de aplicação utilizada pelo usuário.
Utilitário (Utils)	Funções reutilizáveis.

Fonte: Adaptado (Evans 2004, Percival;Gregory 2020)

O funcionamento geral do domínio de um sistema (Figura 20), onde uma lógica de um Utilitário é utilizada por um Caso de Uso, e posteriormente aplicada sobre uma Entidade, que por sua vez acarreta mudança de estado do sistema (ex: atualização de atributos de uma Entidade).

Figura 20 – Representação geral do domínio de um sistema



Fonte: Adaptado (Percival; Gregory, 2020).

Evans (2004), utiliza o conceito de modelagem orientada a domínio na concepção da DS *Domain Driven Design*, popularmente conhecida como DDD. O DDD padroniza a subjetividade da modelagem orientada a domínio, reunindo diversas concepções e suas similaridades, construídas ao longo do tempo. O DDD é dividido em técnicas gráficas de AS, padrões de implementação a nível de código e técnicas de comunicação entre desenvolvedores e especialista do domínio (Gamma *et al.*, 1994).

Segundo Percival e Gregory (2020), é uma evolução da modelagem orientada a domínio, e se alinha com a utilização de *frameworks web*, uma vez que tais ferramentas disponibilizam estruturas pré-construídas para sua utilização, o que facilita o processo de desenvolvimento.

As principais técnicas da modelagem orientada a domínio são:

Linguagem ubíqua: Conjunto de termos e relações de comunicação entre desenvolvedor e agente especialista do domínio do SW. Consiste na definição do vocabulário que será utilizado durante a sua concepção e implementação.

Fronteiras de domínio: Separação entre camadas Apresentação, Lógica e persistência.

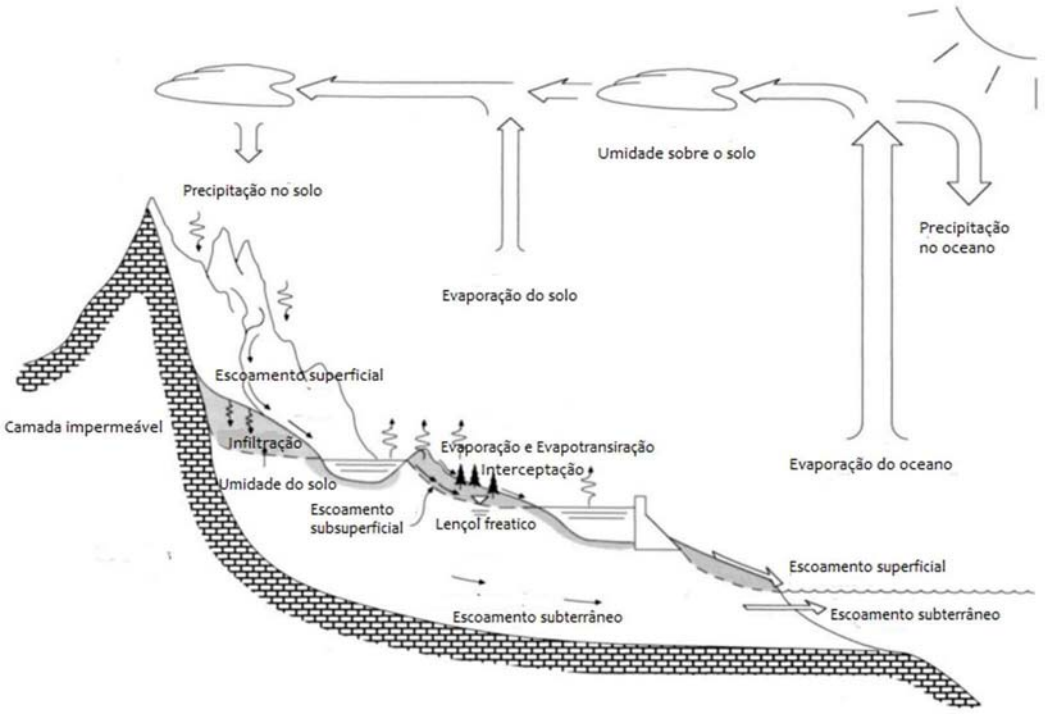
Fronteiras de responsabilidade: Separação de contexto e distribuição de entidades de acordo com sua responsabilidade.

As seções 2.3 e 2.4 apresentam os conceitos empregados no domínio da API REST implementada neste trabalho.

2.3 Ciclo hidrológico, bacia hidrográfica, hidrologia e rede de monitoramento

O ciclo hidrológico é o processo contínuo de circulação da água no globo, abrangendo seus sistemas atmosférico, terrestre e oceânico (Figura 21). Tais sistemas são aquecidos pela incidência de energia solar, provocando a evaporação da água presente na superfície. O vapor d'água é transportado pelo ar na atmosfera, formando as nuvens, onde sob determinadas condições, retorna a superfície terrestre na forma de precipitação (Collischonn; Dornelles, 2015).

Figura 21 - Ciclo Hidrológico em escala de bacia hidrográfica



Fonte: Adaptado Chow, Maidment e Mays (1988)

Os fenômenos hidrológicos são os processos naturais de armazenamento e transporte da água entre as diversas fases do ciclo. Em escala de bacia hidrográfica, a quantificação dos fenômenos hidrológicos pode ser obtida através da sistematização da observação e medição de suas variáveis características (Quadro 4) (Chow; Maidment; Mays, 1988; Santos et al., 2001; Naghettini; Pinto, 2007).

Quadro 4 - Fenômenos hidrológicos e suas variáveis características

Fenômeno	Variável característica	Unidade
Precipitação	Altura Intensidade Duração	mm, cm mm/h h, min
Evaporação/Evapotranspiração	Intensidade Total	mm/dia, mm/mês mm, cm
Interceptação	Altura	mm
Infiltração	Intensidade Altura	mm/h mm, cm
Escoamento sub/superficial	Fluxo Volume Altura Equivalente	l/s, m³/S m³, 10 m³, (m³/s).mês mm/m², cm/m²
Escoamento subterrâneo	Fluxo Volume	l/min, l/h, m³/dia m³, 10 m³

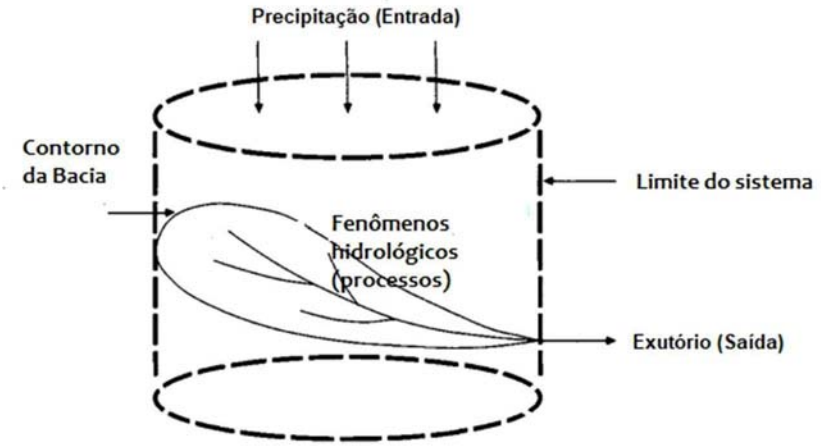
Fonte: Adaptado Naghetinni; Pinto (2007)

Segundo Santos et al. (2011), a bacia hidrográfica de um dado ponto de um rio, é a superfície terrestre delimitada por divisores topográficos originando uma superfície de drenagem, na qual toda água precipitada, não evaporada, infiltrada ou retida, escoa em direção. Tal ponto denomina-se seção de controle de um rio ou exutório da bacia hidrográfica.

Chow, Maidment e Mays (1988) representam uma bacia hidrográfica como um sistema hidráulico (Figura 22). Um sistema hidráulico é um volume no espaço, cercado por um limite (divisores topográficos), do qual recebe água da chuva como principal forma de entrada, operando-as em processos internos do sistema (fenômenos hidrológicos), por fim, produzindo saída (vazão)

.

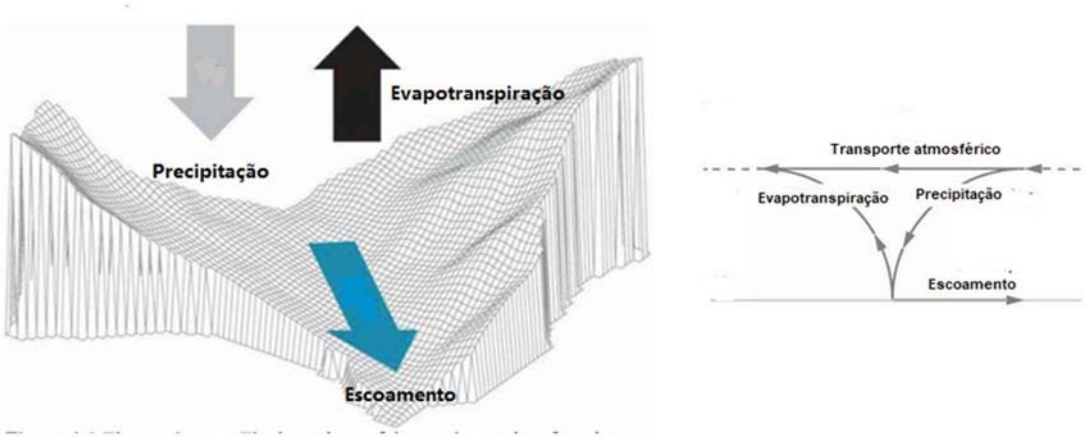
Figura 22 - Representação da bacia hidrográfica como um sistema hidráulico



Fonte: : Adaptado Chow, Maidment e Mays (1988)

Neste contexto, restringindo-se a parte terrestre do ciclo hidrológico (Figura 23), define-se a hidrologia como o estudo da precipitação e escoamento em uma bacia hidrográfica, considerando que a circulação dá água pode ser observada, prevista e simulada (Santo *et al.,* 2001; Clarke; Dias, 2003; Hornberger *et al.*, 2014).

Figura 23 - Representação da parte terrestre do ciclo hidrológico



Fonte: Adaptado Hornberger *et al.* (2014)

Entre os modelos mais simples utilizados na hidrologia, destaca-se a equação do balanço hídrico (Equação 1), que determina a quantidade de água na bacia hidrográfica em determinado momento. Tal equação serve de base para a concepção de modelos mais complexos, como o modelo hidrológico para grandes bacias (MGB), que utiliza dados geoespaciais para representação da bacia hidrográfica, equações conceituais para a representação de fenômenos do ciclo hidrológico, e equações físicas para representação de vazão ao longo da rede de drenagem (Santos *et al.,* Ferreira *et al.*; 2011, Guandique; Morais, 2015; Alves *et al.*, 2020; Farias *et al.*; 2023).

$$P - ET = D + \Delta S \quad (1)$$

Onde:

P = Precipitação;

ET = Evapotranspiração;

D = Deflúvio (escoamento total) ;

ΔS = Variação do armazenamento (superficial e/ou subterrâneo)

A efeitos de simulações e previsões, Santos *et al.* (2001) destaca que, por mais sofisticados sejam os modelos, a representatividade dos resultados é diretamente influenciada pela disponibilidade e qualidade dos dados do fenômeno observado. Desta forma, reduzindo-se ao âmbito das águas superficiais, define-se a hidrometria como o campo técnico-científico presente na hidrologia, com objetivo de sistematizar a obtenção e tratamento de dados básicos, que caracterizam a precipitação e escoamento, através do uso de pontos de observação.

Ponto de Observação (PO) é o local onde se realiza a instalação de estruturas e equipamentos necessários para a medição e registro de dados, caracterizando sua variação no tempo e/ou espaço. Um PO pode ser manual, onde o registro e coleta de dados necessita de um agente coletor-observador, ou automática, onde o registro dos dados é realizado automaticamente ao longo do tempo, de forma analógica ou digital.

Segundo Santos *et al.* (2001) a escolha do local instalação de um PO, deve levar em consideração a representatividade da área, seu acesso, segurança dos equipamentos, meios de comunicação, e interferências antrópicas e/ou naturais.

A distribuição estratégica de diversos PO constitui uma rede de monitoramento, com objetivo prover informações hidrometeorológicas que caracterizam o estado do sistema em qualquer ponto da bacia hidrográfica, no que concerne à atmosfera e os recursos hídricos, mediante o cruzamento coerente dos dados registrados pelos PO que a compõe (Santos *et al.*, 2001; Naghetinni, 2012).

Uma rede de monitoramento hidrometereológico completa, abrange a pluviometria, fluviometria, evapotranspiração, águas subterrâneas, transporte de sedimentos, qualidade de água, e elementos climatológicos (Tabela 3), que são as grandezas atmosféricas que influenciam na intensidade de um fenômeno hidrológico,

consequentemente, o estado do sistema (Naghettini, 2012; Fiorin; Ross, 2015; Ynoue *et al*., 2017).

Tabela 3 – Elementos climáticos

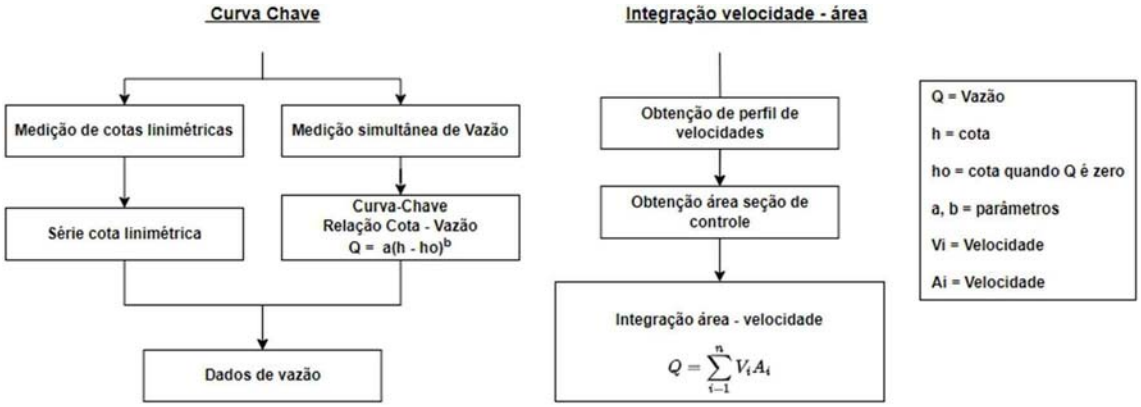
Elemento	Unidade de medida
Temperatura	°C, °F, K
Umidade relativa	%
Umidade absoluta	g/m³
Umidade específica	g/kg
Direção do Vento	Pontos Cardeais
Velocidade do Vento	m/s, km/h
Pressão Atmosférica	mb, hPa
Radiação solar	W/m²

Fonte: Fiorin; Ross (2015)

Os PO fluviométricos são responsáveis pela obtenção de dados de velocidade e nível do curso d'água (cota linimétrica). Os valores de velocidade possibilitam a obtenção da vazão de um rio, através da integração da área da seção de controle e seu perfil de velocidades. Este método é oneroso, necessitando de tempo, equipamentos e mão de obra qualificada, além de condições climáticas favoráveis para medição, comprometendo a frequência de observações.

Com a medição sistemática de vazão em um mesmo local, é possível determinar a relação entre a cota do curso d'água em sua seção e a sua vazão. Desta forma, a obtenção de valores de vazão ocorre de forma indireta, possibilitando maior frequência na obtenção de dados, através da cota observada. Tal relação denomina-se curva-chave, ou curva de descarga (Figura 24) (Santos *et al.,* 2001; Naghettini 2012, Collischon; Dornelles, 2015).

Figura 24 – Diagrama de etapas para a obtenção de valores de vazão.



Fonte: Adaptado Naghettini (2012)

Em síntese, espera-se como produto resultante de uma rede de monitoramento hidrometeorológica, um conjunto ordenado de registros efetuados pelos PO, que caracterizem o comportamento das variáveis hidrometeorológicas ao longo do tempo, e atendam a qualidade exigida para a sua utilização como subsídio nas tomadas de decisão em todos os aspectos e áreas de aplicação relacionadas a gestão de recursos hídricos (Pereira; Barbieiro; Quevedo, 2020).

2.4 Enfrentamento de desastres hidrológicos

Um desastre hidrológico se caracteriza como evento súbito relacionado a água, de origem natural ou antrópica, do qual a ocorrência resulta em danos significativos à sociedade e ao meio ambiente. Dentre tais desastres, destacam-se as secas, inundações, enxurradas e alagamentos. A prevenção e mitigação de tais desastres requer um arcabouço legal que configure um ambiente organizacional colaborativo entre o Estado, entidades privadas e comunidades (Pereira; Barbieiro; Quevedo, 2020; Robaina et al., 2024)

No Brasil, norteiam tal ambiente:

- Lei Federal n° 9.433/1997, conhecida como Lei das Águas, institui a Política Nacional dos Recursos Hídricos (PNRH), cria o Sistema Nacional de

Gerenciamento dos Recursos Hídricos (SINGREH), e define a bacia hidrográfica como unidade territorial para sua implementação e atuação. Entre seus objetivos está a prevenção e defesa contra eventos hidrológicos extremos de origem natural ou decorrente do uso inadequado dos recursos hídricos.

- Lei Federal n°9.984/2000, que institui a criação da Agência Nacional das Águas e Saneamento Básico (ANA), e a define como entidade integrante do SINGREH, atribuindo a responsabilidade pela implementação da PNRH.
- Lei Federal n° 12.608/2012, que institui a Política Nacional de Proteção e Defesa Civil (PNPDEC), e adota a bacia hidrográfica como unidade territorial de análise das ações de prevenção de desastres relacionados a corpos d'água.
- Decreto n° 10.593/2020, que estabelece um conjunto de princípios, diretrizes e objetivos, para o norteamento nas estratégias de gestão de ri e desastres a ser implementada pela União, Estados, Distrito Federal e Municípios (Brasil, 1997, 2012, 2020; Robaina et al., 2024).

Ressalte-se o papel da ANA, que tem entre suas atribuições estão o planejamento e a promoção de ações destinadas a prevenção ou minimização dos efeitos de secas e inundações, em apoio aos Estados e Municípios, em conjunto com a Defesa Civil. Como ferramenta para atingir tal objetivo, a ANA em 2009 inaugurou a Sala de Situação, para monitoramento e análise de eventos hidroclimatológicos, e pontos de observação de interesse da União. Em 2010 deram início a implementações de Salas de Situação estaduais, com atuação em escala reduzida, onde em 2013, em parceria com a Secretaria do Meio Ambiente e Infraestrutura (SEMA), foi criada a sala do Rio Grande do Sul, dando início em sua atuação em 2015, no âmbito de monitoramento e produção de informações, para previsões e prevenção de eventos hidroclimatológicos críticos extremos (ANA, 2024).

Outro órgão de prevenção atuante no Brasil é o Centro Nacional de Monitoramento e Alertas de Desastres Naturais (CEMADEN),foi criado em 2011, sob o contexto de uma sequência de desastres causados por inundações, enchentes e deslizamentos que atingiram anteriormente o país. O CEMADEN é responsável pelo gerenciamento de dados provenientes de radares meteorológicos, pluviômetros e dados provenientes de previsões climáticas, repassando informações aos órgãos

competentes, com objetivo de antecipação perante possibilidade de eventos meteorológicos que possam acarretar em um desastre natural (Cemaden, 2024).

O Serviço Geológico do Brasil - CPRM também atua nas ações relacionadas a ocorrência de desastres naturais. O órgão consta com salas de monitoramento com o objetivo de produção de informações e dados hidrológicos, que subsidiam as tomadas de decisões por parte de órgãos responsáveis pela mitigação dos efeitos de eventos hidrológicos extremos.

Pereira, Barbiero e Quevedo (2020) destacam a importância do monitoramento hidrológico em tempo hábil para a produção de informações que auxiliem na identificação de riscos iminentes e proporcionem meios para escolhas assertivas dos tomadores de decisão. Kobiyama et al. (2006) enfatizam o monitoramento contínuo de processos do ciclo hidrológico e suas variáveis, em especial a precipitação e eventos de chuvas extremas, como atividade fundamental para a elaboração de um sistema de alerta e prevenção de desastres naturais (Quadro 5)

Quadro 5 - Fragmento da classificação dos desastres naturais segundo a Classificação e Codificação Brasileira de desastras (COBRADE)

Classificação		
	Grupo	**Subgrupo**
Desastres Naturais	Hidrológicos	Inundações Enxurradas Alagamentos
	Meteorológico	Sistemas de Grande Escala/Regional Tempestade Tempestade Extremas
	Climatológico	Temperaturas Extremas Seca Estiagem
	Geológico	Deslizamento

Fonte: Adaptado Robaina et al. (2024)

Kobiyama et al. (2009), classifica dois tipos de medidas preventivas básicas historicamente utilizadas no Brasil, no tocante ao enfrentamento de desastres hidrológicos, sendo elas: medidas estruturais e não-estruturais. As medidas estruturais envolvem obras de engenharia, como a construção de barragens, diques, alargamento de rios, reflorestamento, implementação de rede de monitoramento, concepção e implementação de sistema de alerta entre outros. As medidas não estruturais envolvem ações de planejamento, gerenciamento, zoneamento de área crítica, conscientização da população.

O autor ainda destaca três etapas de atuação, nomeando-as como pré-desastre, desastre e pós-desastre, definindo de modo geral, o objetivo das ações a serem tomadas em cada uma delas. Corroborando com tais divisões, Robaina et al. (2024) as relaciona com ações e diretrizes previstas no Decreto n° 10.593/20 conforme informações contidas no Quadro 6 .

Quadro 6 – Etapas de prevenção a desastres naturais

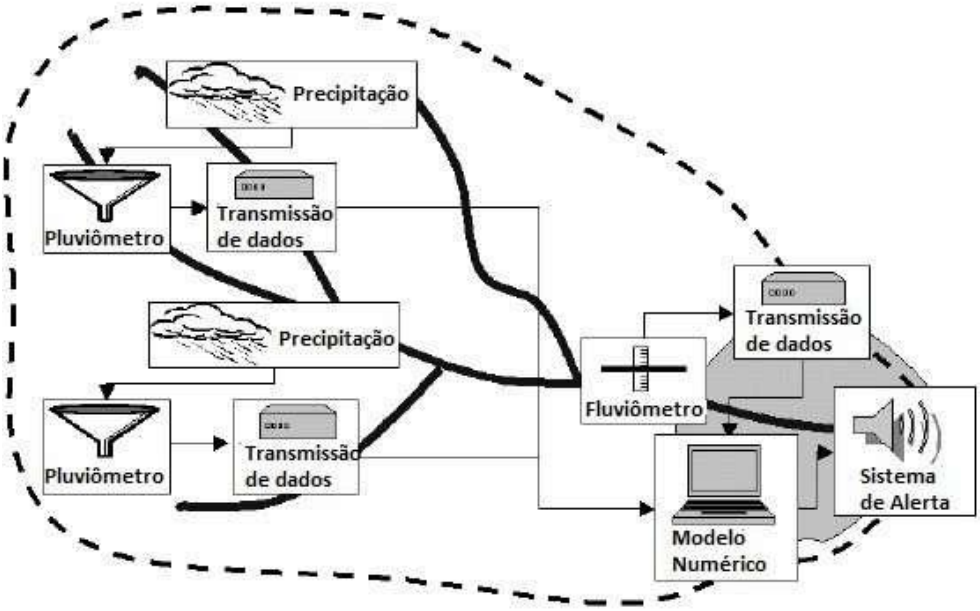
Etapa	Descrição	Ações Decreto N° 10.593/20
Pré-desastre	**Antes do evento ocorrer**	**Mitigação, Prevenção, Preparação,**
	Atividades e estudos baseados em eventos anteriores Coleta de relatos e informações com a comunidade Planejamento para otimização das ações de Resposta	
Desastre	**Durante a ocorrência**	**Resposta, Restabelecimento**
	Ações emergenciais baseadas em informações confiáveis Acompanhamento, Monitoramento, e Divulgação de informações	
Pós-desastre	**Após a ocorrência**	**Restabelecimento, Recuperação**
	Restauração e/ou reconstrução Levantamento e organização de informações	

Fonte: Adaptado (Kobiyama et al. 2009; Brasil 2020; Robaina et al., 2024)

A Figura 25 ilustra a dinâmica de um sistema de alerta em uma bacia hidrográfica e um fluxograma de visão geral, com seus principais componentes sendo o monitoramento, a transmissão de dados, zoneamento de área de riso, modelagem, simulação, divulgação de informações paras as autoridades e instituições responsáveis para emissão de alertas e avisos a população (Quadro 7).

.

Figura 25 - Representação sistema de alerta para desastres hidrológicos



Fonte: Adaptado (Kobiyama et al., 2006)

Quadro 7 – Levantamento de ações propostas para a prevenção e mitigação de eventos de inundação.
Fonte: Adaptado (Kobiyama et al., 2006; Santos; Matos, 2021; Robaina et al., 2024)

A linguagem de programação Python e o *Django* tem sido utilizada na criação de sistemas de alerta contra inundações. Ricardo *et al.* (2024), utilizam a linguagem e para a criação de sistema de simulação de inundação no rio Águeda em Portugal. O sistema monitora a cota de limiar de inundação, realizando o envio de mensagens de alerta quando atingida e se conecta com banco de dados de terceiros, para a aquisição de dados para a realização das simulações de inundação.

Lima, Freiman e Camboim (2022), utilizam o *Django* para a criação de um SW de mapeamento colaborativo de áreas de de alagamento no município de Nova Friburgo no estado do Rio de Janeiro. O sistema aceita o cadastro de usuários colaboradores, dos quais são permitidos o envio de informações relativas à ocorrência de alagamentos, deslizamentos e inundações.

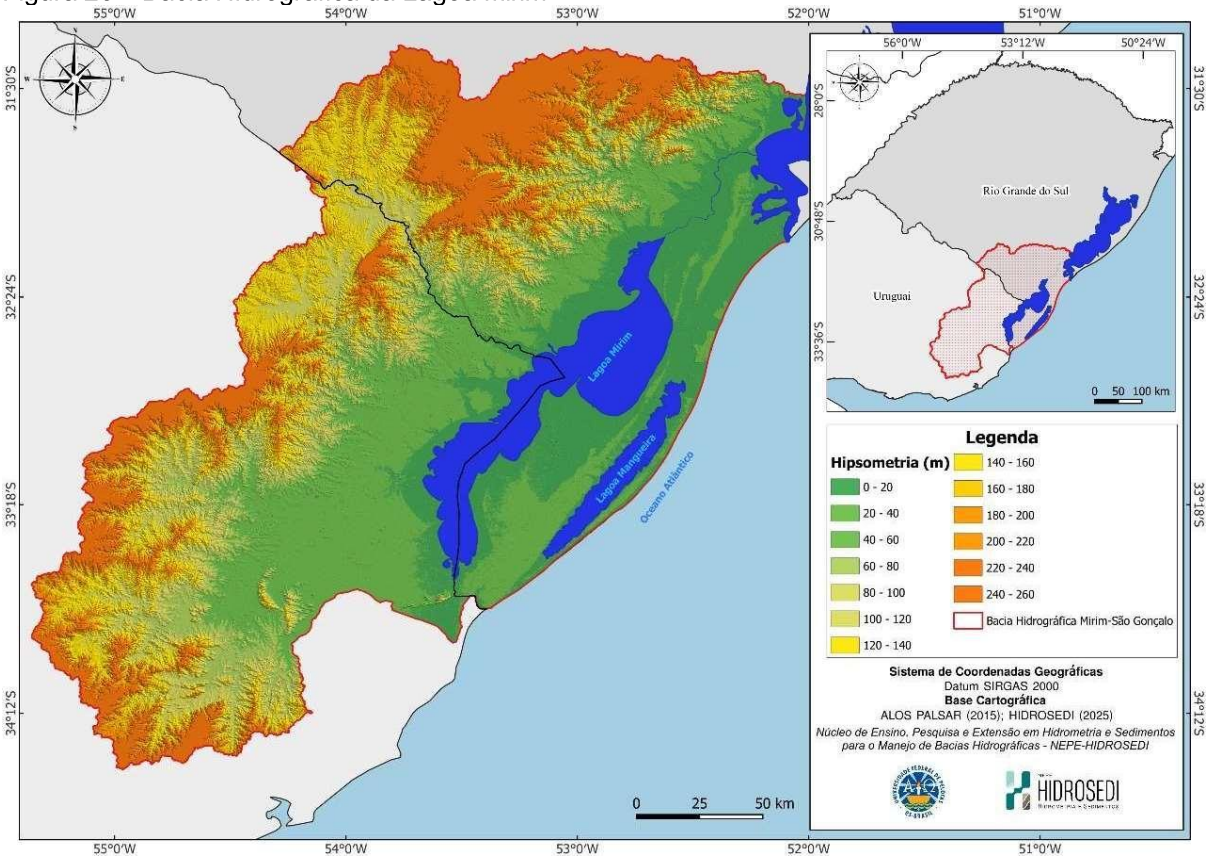
Duarte et al. (2023) desenvolveram um sistema de monitoramento em tempo real da chuva e da vazão na Bacia do Rio Maria Santa da Vitória, no estado do Espirito Santo. O sistema utiliza o *Django*, e tem por objetivo a disponibilização de previsão de informações hidrológicas obtidas através do modelo MGB-IPH, tendo como entrada dados de chuva.

3 Materiais e métodos

3.1 Área de estudo

A Bacia Hidrográfica Mirim-São Gonçalo (BHMSG) situa-se na região sul do Rio Grande do Sul, entre os paralelos 31°30’ e 34° 30’ S e os meridianos 52° e 56° W, com área aproximada de 62.250 km², sendo 29.250 km² em território nacional e 33.000 em território uruguaio (Figura 26). O canal São Gonçalo, corpo hídrico presente na BHMSG, com aproximadamente 76 km de extensão, 250 metros de largura e 5 metros de profundidade média, com pontos atingindo profundidade de até 15 metros, é responsável pela conexão entre a Lagoa Mirim e a Laguna dos Patos, esta qual estabelece ligação direta com o Oceano Atlântico. O canal abrange os municípios de Pelotas, Rio Grande, Capão do Leão e Arroio Grande, exercendo grande importância socioeconômica e ambiental para a região.

Figura 26 – Bacia Hidrográfica da Lagoa Mirim



Fonte: Núcleo de Pesquisa Ensino e Extensão em Hidrometria e Sedimentos para o manejo de Bacias Hidrográficas - NEPE-HidroSedi.

3.2 Revisão bibliográfica Sistemática

Segundo Botelho, Cunha e Macedo (2011) o processo de Revisão Bibliográfica Sistemática (RBS) consiste em definir um determinado tema de pesquisa e identificar trabalhos relevantes em diferentes fontes de dados (artigos, livros, dissertações, teses, relatórios técnicos, entre outros). Os trabalhos identificados, devem ser revisados e sintetizados levando em consideração sua reprodutibilidade. Os autores definem como objetivo principal de uma RBS, reunir o estado conhecido sobre

determinado tema, identificar lacunas e fornecer um direcionamento para possíveis pesquisas futuras.

Gomes e Caminha (2014), defendem que a produção de trabalhos realizados com o método de RBS, podem exercer o papel de ponto de partida sobre determinada área de estudo, dando uma perspectiva substancial sobre a rede de pensamentos e conceitos construídos, testados e validados.

No tocante a área hidro ambiental, Almeida (2024), utiliza a RBS para o levantamento de técnicas de recuperação ambiental em áreas secas degradadas, dando ênfase na importância nos critérios de escolha de aplicação em suas conclusões. Eustáquio e Rossoni (2022), relatam a eficácia da padronização do processo de tratamento de águas residuárias, apontando os gargalos identificados nos trabalhos selecionados e questões de aspecto econômicos na redução de custos. Ruezzene, Miranda, Tech e Mauad (2020), utilizam a RBS para um levantamento de técnicas empregadas no preenchimento de falhas de dados de precipitação, tendo como resultado um quadro para consultas especificando sua descrição, motivos de utilização, vantagens e desvantagens dos métodos utilizados.

Neste trabalho a RBS considerada é a abordagem integrativa, descrita pelo levantamento de diferentes métodos de revisões realizado pela Faculdade de Ciências Agronômicas da UNESP – Campus de Botucatu . Neste contexto, o método RBS-Integrativo considera a revisão com a combinação de trabalhos publicados com dados empíricos e teóricos (Quadro 8). Tais trabalhos podem ser direcionados a revisão e definição de conceitos, identificação de lacunas e oportunidades de continuação e/ou novas propostas de estudos em determinada área (Unesp, 2015).

Quadro 8 – Passos a serem empregados na metodologia RBS-Integrativa.

Passo	Ação	Resultado
1	Formulação de questão de investigação.	Qual a questão central a ser investigada ?
		Existe mais de uma área de estudo envolvida ? Quais ?
2	Produção de protocolo de investigação e/ou registro.	Quais os critérios eliminatórios?
3	Categorização dos trabalhos selecionados.	Quais as categorias notáveis ? se necessário.
4	Descrição das fontes de pesquisa	Identificação das fontes de dados consultadas.
5	Descrição e discussão dos resultados.	Apontamento de resultados, lacunas e/ou possibilidades de inovações.

Fonte 1 – Adaptado (Unesp, 2015).

Foi realizado o método de RBS-Integrativa, realizando as respostas propostas no Quadro 8. As respostas são apresentadas abaixo:

Qual a questão central a ser investigada ?
A implementação de uma API REST utilizando modelagem orientada a domínio, considerando como agente especialista do domínio do sistema, engenheiros especializados em recursos hídricos.

Existe mais de uma área de estudo envolvida ?
Sim, sendo elas desenvolvimento SW, DS, Hidrologia, Hidrometria, Climatologia e Prevenção e mitigação de desastres naturais de natureza hídrica.

Quais os critérios eliminatórios ?
Os trabalhos selecionados passaram por critério de eliminação de acordo com os temas da pesquisa. A primeira etapa consistia em uma análise geral. Em livros, foi realizada leituras em capítulos pertinentes a cada tema. Em artigos, trabalhos de conclusão de curso, dissertações e teses, a leitura inicial foi resumo e resultados e discussões. De acordo com os resultados, posteriormente a leitura total do artigo era realizada. Os critérios de eliminação eram a discrepância entre os temas escolhidos, de acordo com a tecnologia empregada, tema e resultados apresentados.

Quais as fontes de pesquisa consultadas?
As fontes de pesquisa consultadas foram determinadas de acordo com as áreas estabelecidas. Para Hidrologia, foram consultadas a bibliografia recomendada nas disciplinas de Hidrometria Aplicada, Fundamentos de Hidrologia, Engenharia Hidrológica, Climatologia e Climatologia Agrícola, dos cursos de Pós-graduação em Recursos Hídricos, Engenharia Hídrica e Engenharia Agrícola, da Universidade Federal de Pelotas. Para as áreas de SW, DS, foram consultadas a bibliografia básica recomendada nos cursos de Projeto de Sistema Web e Programador Web, oferecidos pelo Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul, e material utilizado como referência nas apresentações dos palestrantes de diversas edições da Python Brasil. Para a area de Prevenção e mitigação de desastres naturais de natureza hídrica, foram consultadas o material de referências apresentados em

palestras realizadas de forma online pela Associação Brasileira de Recursos Hídricos e ANA, por meio da iniciativa Hidrotalks e Webinar.

3.3 Concepção e implementação da API REST

3.3.1 Ambiente de desenvolvimento e versionamento de código.

O versionamento de código foi realizado através das ferramentas *git* e e *github*. Para garantir a consistência e rastreabilidade do versionamento, foi realizado a padronização dos *commits*, seguindo convenção adotada pela comunidade de desenvolvedores DRF (Quadro 9).

Quadro 9 – Padronização de commits adotada durante o desenvolvimento.

Commit	Descrição
init	Commit inicial do projeto.
lib	Adiciona nova biblioteca
feat	Nova funcionalidade para o usuário.
fix	Nova correção de bug para o usuário.
docs	Mudança na documentação.
style	Formatação, lint, falta de virgula, entre outros, não atualiza o codigo em produção.
refactor	Refatoração de código em produção, ex.: renomeação de variáveis.
test	Adiciona testes, refatora tests; sem mudança em produção.
chore	Atualização de tarefas, etapa de desenvolvimento; não atualiza código em produção.

Fonte : Autor

A implementação seguiu através de um ambiente virtual (Quadro 10), com a utilização da biblioteca padrão Python *virtualenv.* Um ambiente virtual isolado permite o gerenciamento de dependências de forma controlada, facilitando o versionamento de pacotes. Desta forma, o uso de pacotes instalados de forma global é

desnecessário, tendo cada projeto um ambiente virtual isolado, evitando conflitos entre diferentes projetos

Quadro 10 – Bibliotecas para a configuração de ambiente virtual de desenvolvimento

Biblioteca	Descrição
pre-commit	automatiza a verificação de código, como a formatação antes de salvar a versão.
isort	organiza automaticamente as importações realizadas em um arquivo .py
black	aplica a formatação do código seguindo regras pré-estabelecidas.
autopep8	aplica a formatação do código seguindo regras de convenção pep8

Fonte : Autor

3.3.2 Definição dos requisitos do sistema

A implementação de API-REST em redes de monitoramento regionais manuseadas por entidades não ligadas a ANA, tem por objetivo fornecer dados estruturados a respeito do monitoramento hidrometeorológico da região. Tal atividade pode suprir a carência de informações em tempo real durante a ocorrência de eventos críticos, de regiões afastas de grandes centros urbanos. A implementação pode ser realizada de diversas formas, de modo mais geral, a utilização de frameworks web, como o FastAPI, Flask e Django REST framework (DRF).

Flask é um *framework* que visa a customização máxima pelo usuário, não inclui funcionalidades como autenticação, ferramentas de manipulação de dados em registro no banco de dados, exige conhecimentos avançado das bibliotecas do ecossistema Python e padrões de projeto a nível de código. FastAPI é um *framework* com foco em APIs baseadas em *async/await* , consta com documentação automática e interativa, sendo ideais para sistemas com logica de interação com usuário, como chatbots e demais integrações em tempo real.

A implementação foi realizada com a utilização de DRF. Os critérios de escolha foram sua popularidade, o que permite maior suporte com comunidade de usuários durante a resolução de problemas, e seu ecossistema de bibliotecas, que auxiliam durante o desenvolvimento. Os critérios utilizados em consideração na construção do DS são apresentados no Quadro 11.

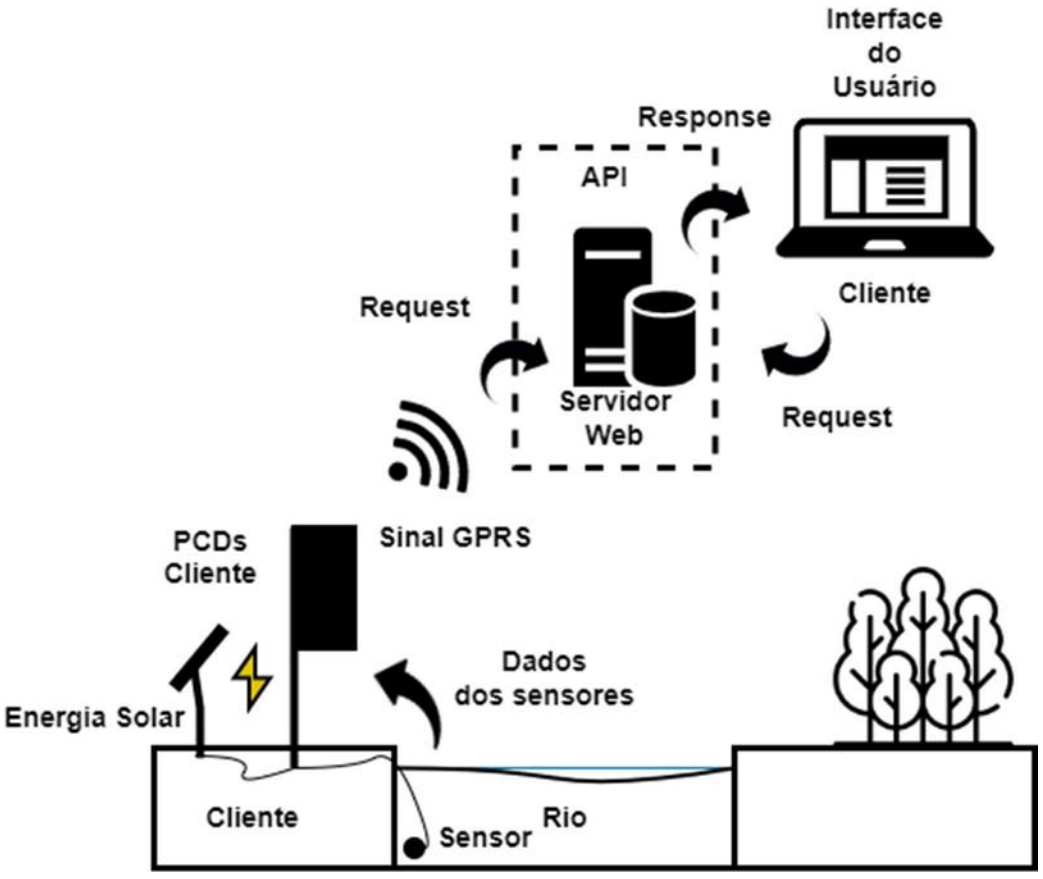
Quadro 11 – Caraterísticas da AS

Aspecto	Arquitetura
Foco	Visão geral do funcionamento do sistema.
Nível de Detalhe	Amplo e alto nível de generalizado.
Escopo	Interações entre camadas de alto nível do SW.
Granularidade	Componentes de alto nível de abstração .
Resultado	Diagrama com estrutura geral de funcionamento.

Fonte: Adaptado (Stair; Reynalds, 2016; Marcoskalinowski *et al.*, 2023; Wilson, 2024)

A Figura 27 considera uma Plataforma de coleta de dados em funcionamento, que realiza uma requisição *POST a* API REST , desta forma registrando os dados hidrometereologicos em banco de dados, possibilitando acesso através de SW.

Figura 27 Representação da arquitetura do sistema em alto nível de abstração.



Fonte: Autor

3.3.3 Definição do domínio do sistema

A definição da linguagem ubíqua para a implementação da API, levou em consideração as estruturas fornecidas pelo *Django e DRF* (Quadro 12). Em relação as *urls*, levou-se em consideração a convenção adotada pela comunidade de desenvolvedores DRF. A convenção adota o nome *url* para endereços que direcionam as operações CRUD para os recursos do sistema, e *endpoint* para endereços que direcionam para a utilização de Casos de uso

Quadro 12 – Linguagem ubíqua levando em consideração Django e DRF,

Elemento de Domínio	Contextualização Django - Django REST framework
Entidade	Qualquer objeto presente no domínio. Configurações, urls, Views, Modelos, entre outros.
Recurso (Resources)	Modelo. Classe representativa, somente com características, registradas em banco de dados.
Caso de Uso (Use Case)	Lógica de aplicação. Função ou classe que faz uso de utilitário e/ou recursos do sistema.
Utilitário (Utils)	Funções Implementadas reutilizáveis em Casos de Uso.
Url	Endereço direcionado as operações CRUD aplicadas sobre os Recursos.
Endpoints	Endereço direcionado a utilização de Casos de uso.

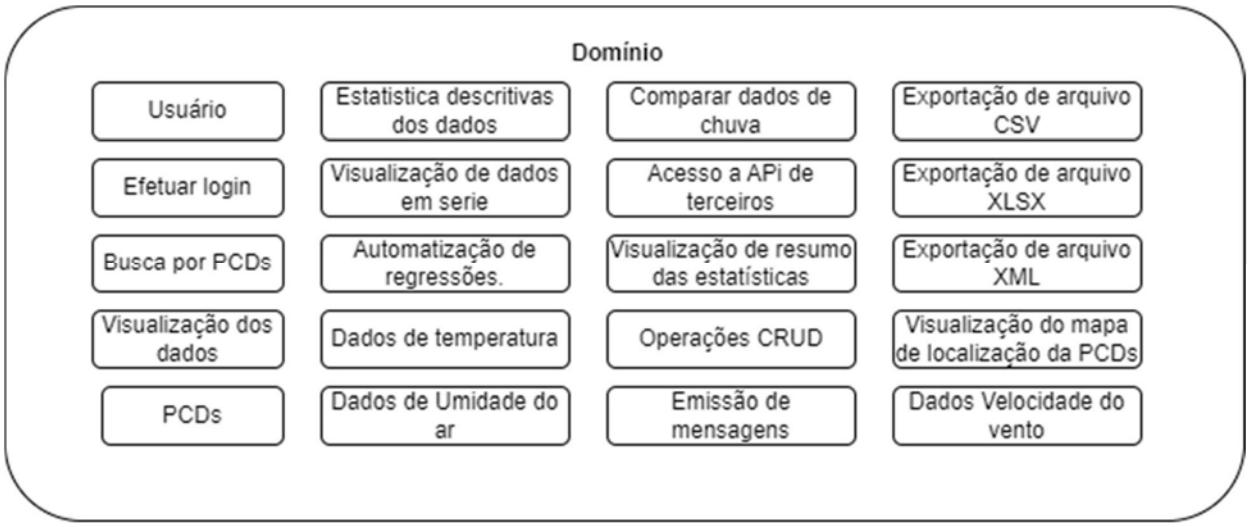
Fonte: Autor.

Nesta etapa foi realizada a concepção de diagrama inicial do domínio do sistema, pressupondo as diversas possibilidades de uso pelo usuário, considerando um SW com suas três camadas finalizadas, ou seja, sistema *front-end* responsável por disponibilizar o acesso os dados da API, sistema *back-end* responsável pela lógica e persistência dos dados do sistema (Figura 27). Ressalte-se que nem toda entidade adicionada no domínio durante esta etapa inicial de modelagem do domínio necessariamente é implementada no sistema final. Esta abordagem inicial, funciona como um mapa mental, onde toda possibilidade de entidade do sistema, levando em consideração o especialista do domínio e desenvolvedores é bem-vinda.

O levantamento de requisitos com entrevista com terceiros não foi realizado. Os requisitos do sistema foram adicionados de forma subjetiva, levando em consideração a visão de um engenheiro exercendo papel em pesquisa relacionada ao

monitoramento hidrometeorológico, o combate aos desastres naturais de natureza hídrica e o desenvolvimento de sistema web.

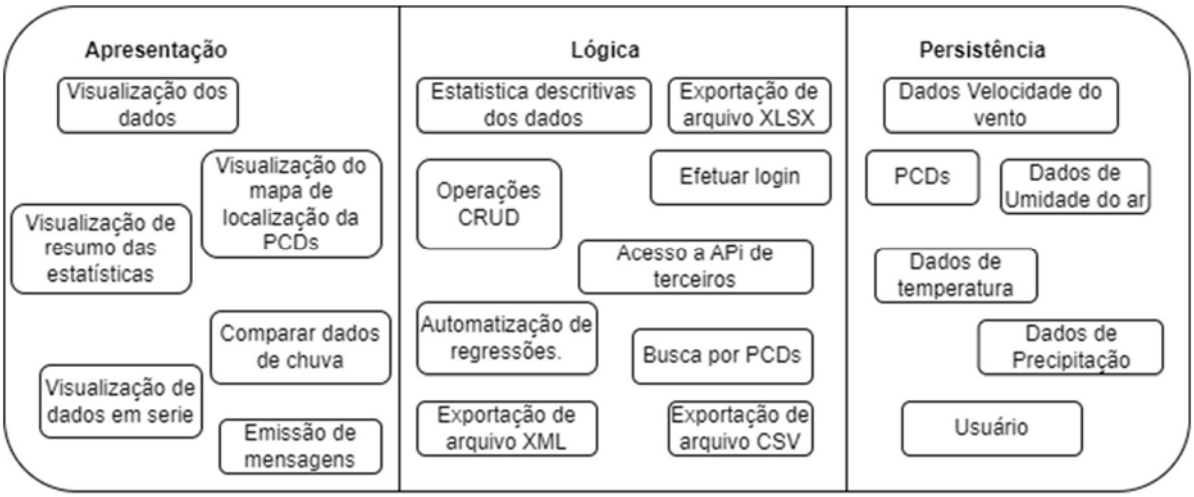
Figura 28 – Entidades do domínio do sistema



Fonte: Autor

Após a definição inicial, o domínio foi refinado com o estabelecimento de fronteiras de responsabilidade (Figura 29). As fronteiras escolhidas foram as apresentadas no item 2.1, Apresentação, Lógica e Persistência. Neste contexto, a camada de apresentação assume papel do componente *front-end.*

Figura 29 – Fronteiras do domínio nas entre as camadas de Apresentação, Lógica e Persistência

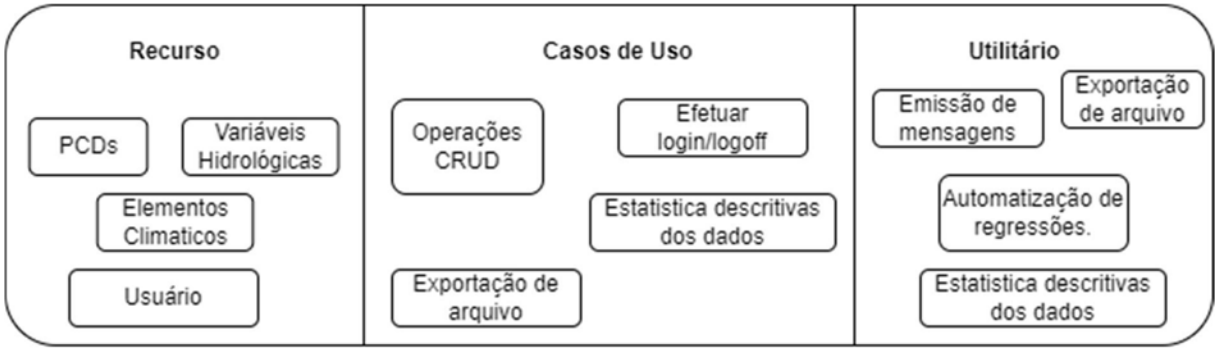


Fonte: Autor

Definidas as fronteiras de responsabilidade e a linguagem ubíqua defino do Quadro 12, foi realizado a definição dos elementos de domínio levando em consideração somente a camada de *back-end* (Figura 30). Neste contexto, a camada de apresentação assume o papel realizado pelas *Views* fornecidas pelo DRF, ou seja,

não leva em consideração a camada *front-end*, e os elementos do domínio são reduzidos levando de acordo os contextos descritos.

Figura 30 – Domínio reduzido ao contexto Django e DRF.



Fonte: Autor.

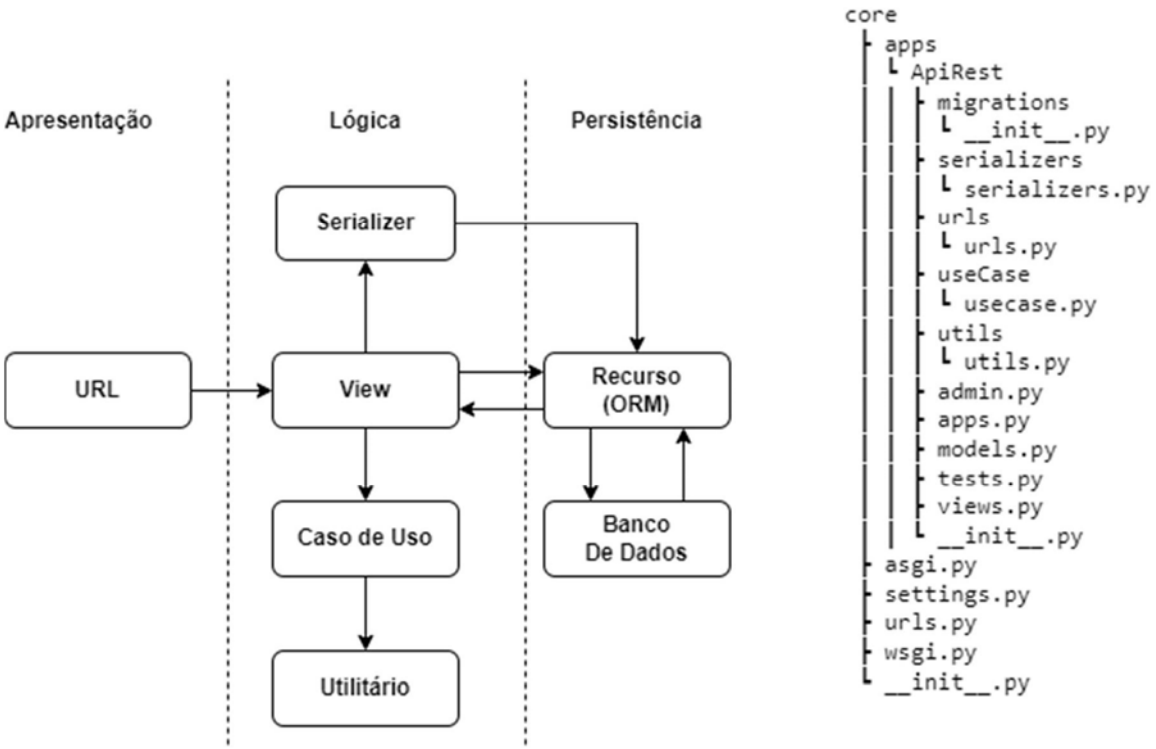
3.3.4 Implementação

A implementação foi realizada seguindo a AS originada da modelagem orientada a domínio realizada, conforme item 3.3.3 e Quadro 12. A AS foi dividida em três camadas para um sistema *web* convencional, Apresentação, Lógica e Persistência (Figura 1). Foram listadas 7 categorias de entidade principais, sendo 5 delas presentes na arquitetura original *DRF, Url, Serializer, View,* Modelo (Recurso) e Banco de dados. E 2 categorias de entidade originarias da modelagem orientada a domínio, Utilitário e Casos de Uso.

O funcionamento da AS segue o seguinte fluxo (Figura 31): O usuário, através de uma url realiza uma requisição a API REST no servidor, por sua vez, a View é acionada e acessa o Serializer, que acessa o Recurso. Se existir lógica de aplicação além das operações CRUD, a View acessa Casos de Uso, que por sua vez tem acesso ao Utilitário

.

Figura 31 – AS originada após a aplicação da modelagem orientada a domínio considerando contexto Django e DRF.



Fonte: Autor

Após definição da arquitetura realizou-se a construção a definição dos recursos disponíveis na *API REST,* sendo eles Usuário, PlataformaColetaDados, PCDLeitura, PCDSensor, PCDLeituraSensor (Quadro 13).

Quadro 13 – Recursos disponíveis na API REST.

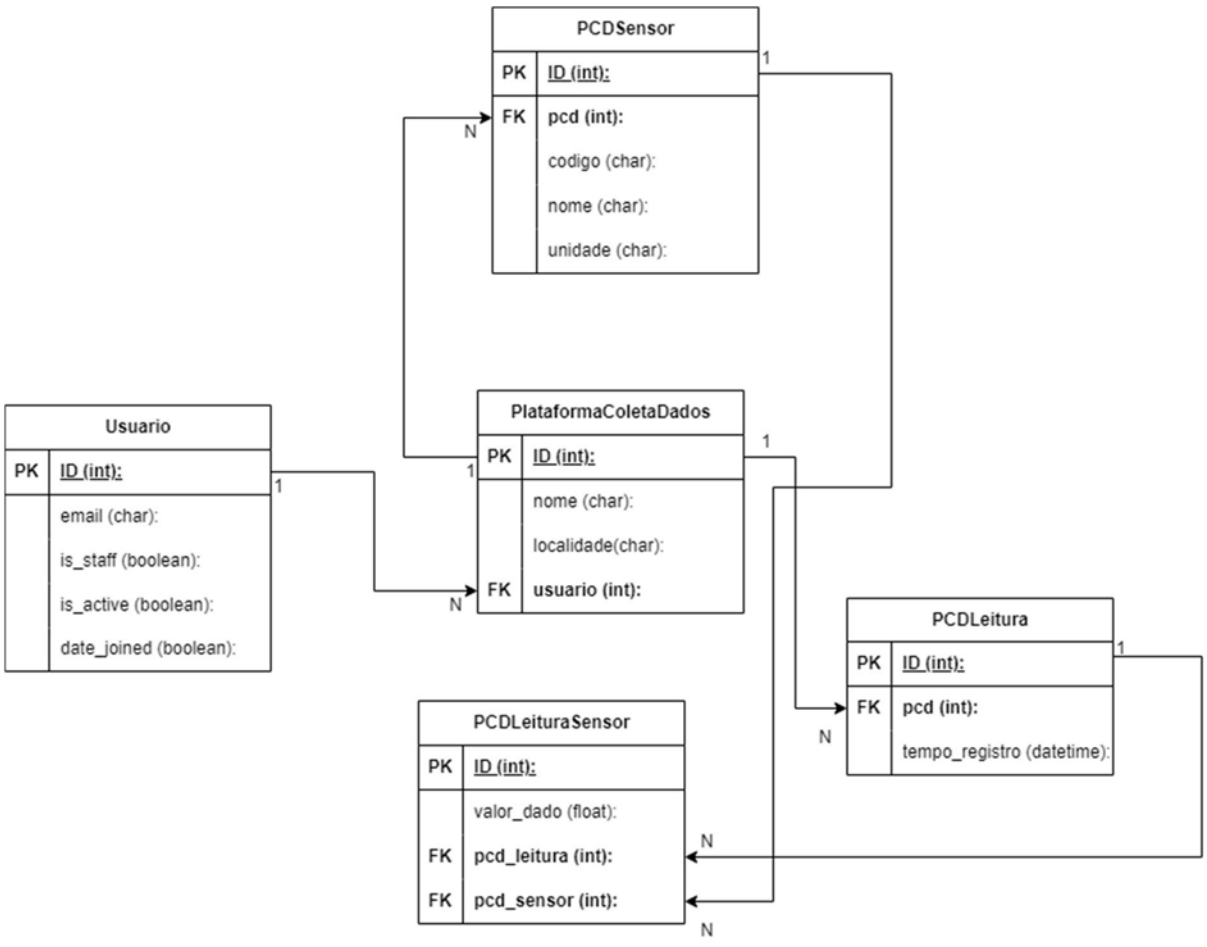
Recurso	Descrição
Usuário	Usuário, agente que interage com o sistema.
PlataformaColetaDados	PCDs responsável pela requisição de registro de dados em determinado intervalo de tempo.
PCDLeitura	Leitura do tipo *datetime* informando quando foi realizado a leitura.
PCDSensor	Tipo de sensor hidroclimatológico associado a PCDs
PCDLeituraSensor	Valor numérico indicado pela leitura do sensor

Fonte: Autor

Os recursos expressam as seguintes dinâmica de relação (Figura 32): Um Usuário pode gerenciar várias PlataformaColetaDados, mas cada plataforma pertence

apena a um Usuário, indicando uma relação um para muitos (1-N). Cada PlataformaColetaDados pode ter múltiplos PCDSensor e pode realizar várias instâncias de PCDLeitura ao longo do tempo. Capa PCDLeitura está associada a uma única PlataformaColetaDados, mas pode conter diversos valores provenientes de PCDSensor diferentes. PCDLeituraSensor exerce o papel de uma tabela de relacionamento que armazena os valores coletados para cada PCDSensor em uma determinada PCDLeitura (Quadro 14).

Figura 32 – Diagrama Entidade de Relacionamento entre os recursos presentes na API REST.



Fonte: Autor.

Quadro 14 – Descrição das entidades e ruas relações.

Entidades	Relação	Descrição
Usuário - PlataformaColetaDados	1 - N	Um objeto Usuário pode estar associado a N objetos PlataformaColetaDados. Por outro lado, cada objeto PlataformaColetaDados pode estar associado somente a um objeto Usuário.
PlataformaColetaDados - PCDSensor	1 - N	Um objeto PlataformaColetaDados pode estar associado a N objetos PCDSensor. Por outro lado, cada objeto PCDSensor pode estar associado somente a um objeto PlataformaColetaDados.
PlataformaColetaDados - PCDLeitura	1 - N	Um objeto PlataformaColetaDados pode estar associado a N objetos PCDLeitura, porém, cada objeto PCDLeitura associa-se apenas a um objeto PlataformaColetaDados.
PCDLeitura - PCDLeituraSensor	1 - N	Um objeto PCDLeitura pode estar associado a N objetos PCDLeituraSensor. Por outro lado, um objeto PCDLeituraSensor associa-se apenas a um objeto PCDLeitura.
PCDSensor - PCDLeituraSensor	1 - N	Um objeto PCDSensor pode estar associado a N objetos PCDLeituraSensor. Por outro lado, um objeto PCDLeituraSensor pode associar-se somente apenas a um objeto PCDSensor.

Fonte: Autor.

Após a definição das relações entre os recursos da API REST, foram implementadas duas aplicações ContasUsuario e *HidroSediDataDash*. A aplicação ContasUsuario é responsável ela criação de três diferentes tipos de usuário com diferentes permissões e níveis de acesso as funcionalidades da API REST, sendo eles: Usuário comum (*standard),* Usuário funcionário (*staff) e* o Usuário mestre (*master)*.

O Usuário comum (*standard)* possui permissão apenas de entrar utilizar a operação *READ. O* Usuário funcionário (*staff),* que possui permissão para registrar PCDs para o Usuário comum, desativar/reativar contas de usuário e realização de operações *Create, Read e Update.* Por último o Usuário mestre (*master),* tem acesso total as funcionalidades do sistema podendo realizar todas as operações *CRUD*.

A aplicação *HidroSediDataDash,* é responsável pela criação dos recursos que compõem o contexto de uma PCDs na API REST, ou seja, ela é responsável pela criação das instâncias dos recursos. Sua principal funcionalidade é são as operações *CREATE*, *READ* e as lógicas intermediarias entre a requisição e a resposta. O seu principal retorno é a estrutura *JSON* organizada de modo que os dados possam ser

preparados para consumo direto em sistemas *front-end,* neste contexto, é onde aplica-se o conceito de modelagem orientada a objetos Casos de Uso.

4 Resultados e di

4.1 Revisão bibliográfica sistemática integrada

Dentre a bibliografia consultada utilizada nesta dissertação, a utilização do método RBS-Integrativa resultaram na reunião dos trabalhados separados em áreas de atuação, que segundo sua conjuntura, resulta na base necessária para a construção de uma API REST, considerando a modelagem orientada a domínio.

Os livros resultantes da RBS-Integrativa (Quadro 15), as obras onde se indica toda sua leitura, referem-se à introdução a conceitos uteis de POO ou uma forte base sobre arquitetura REST, *Django* e *Django REST framework.*

Quadro 15 – Resultado da RBS - Integrativa nas áreas SC SW e DS.

Área de Estudo - SCSW, AS, DS - Livro		
Referência	**Fonte**	**Capítulos**
Stair e Reynolds (2016)	Bibliografia recomendada do curso Programador Web – IFSUL.	1, 3, 7, 12, 13
	Bibliografia recomendada do curso Projeto de Sistema Web – IFSUL.	
Wilson (2024)	Bibliografia recomendada do curso Design Patterns com Python – Udemy.	Todos
Percival e Gregory (2020)	Bibliografia recomendada palestra sobre API REST com DDD - Python Brasil.	1, 2, 3, 4, 5, 6, 7
Lott e Philips (2021)	Bibliografia recomendada curso Python para engenheiro e cientistas – Udemy.	Todos
Hillar (2018)	Bibliografia recomendada curso API REST com Django e Django REST framework – Udemy.	Todos
Vicent (2022)	Bibliografia recomendada curso API REST com Django e Django REST framework – Udemy.	Todos
Brock e McKean (2002)	Citado por Percival e Gregory (2020).	1, 2, 3, 4, 5

Fonte: Autor.

Uma dissertação e um artigo apresentando a construção de aplicações web. O foco de tais trabalhos se baseia na demonstração de técnicas de AS. Os demais

trabalhos citados nesta dissertação, não apresentavam o uso de qualquer metodologia de desenvolvimento. Nenhum dos trabalhos citados apresentaram acesso ao código fonte

Quadro 16 - Resultado da RBS - Integrativa nas áreas SC SW, AS e DS.

Área de Estudo - SW, AS, DS - Artigo, Trabalho de conclusão de curso, Dissertação e Tese	
Referência	**Fonte**
Moreira (2024)	Dissertação Mestrado em Engenharia em Informática e Técnologia web - Universidade Aberta.
Lima, Freiman e Camboim (2022)	Revista Geociências do nordeste.

Fonte: Autor.

Em relação à introdução de conceitos relevantes sobre Hidrologia, Hidrometria e Climatologia, o Quadro 17 apresenta uma os livros fundamentados para a construção do contexto de domínio da API REST.

Quadro 17 - Resultado da RBS - Integrativa nas áreas Hidrologia, Hidrometria e Climatologia.

Área de Estudo - Hidrologia, Hidrometria e Climatologia - Livro		
Referência	**Fonte**	**Capítulos**
Collischonn e Dornelles (2015)	Bibliografia recomendada disciplina Hidrologia do curso de Engenharia Agrícola - UFPEL.	1, 2, 3, 5, 6, 13, 14
Chow, Maidment e Mays (1988)	Bibliografia recomendada disciplina Engenharia Hidrológica do Programa de Pós-Graduação em Recursos Hídricos - UFPEL.	1, 2, 6
Santos (2001)	Bibliografia recomendada disciplina de Hidrometria do curso de Engenharia Hídrica - UFPEL.	1, 2, 4, 5
Naghettini e Pinto (2007)	Bibliografia recomendada disciplina Engenharia Hidrológica do Programa de Pós-Graduação em Recursos Hídricos - UFPEL.	1, 2, 3
Hornberger (2014)	Google Acadêmico	1, 2
Fiorin e Ross (2015)	Bibliografia recomendada disciplina de Climatologia Agrícola do curso de Engenharia Agrícola - UFPEL.	Todos

Fonte: Autor.

O Quadro 18 apresenta os livros resultantes da RBS-Integrativa. Referem-se à introdução de conceitos relevantes sobre Prevenção e mitigação de desastre naturais de natureza hídrica.

Quadro 18 - Resultado da RBS - Integrativa na área de Prevenção e mitigação de desastres naturais de natureza hídricas.

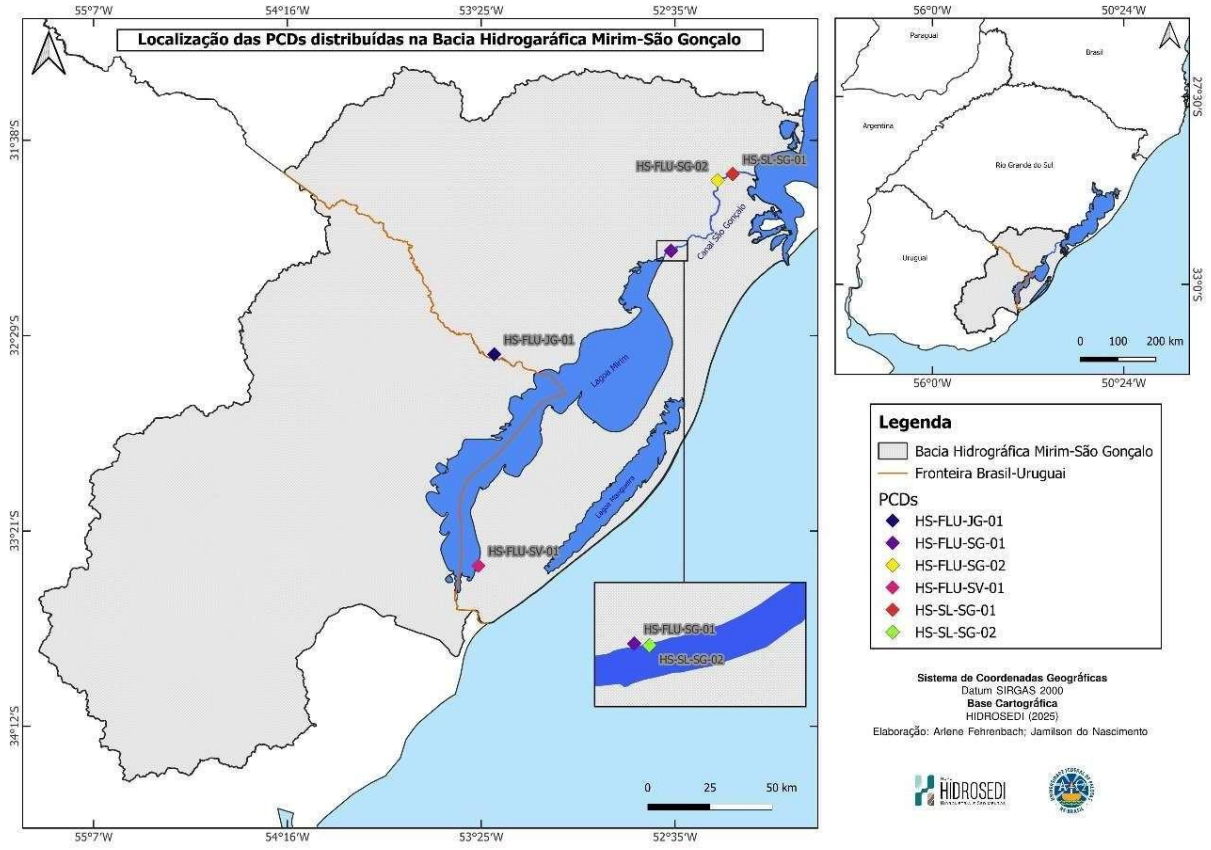
Área de Estudo - Prevenção e mitigação de desastres naturais de natureza hídricas - Livro		
Referência	**Fonte**	**Capítulos**
Robaina (2024)	Google Acadêmico.	1, 2, 3
Kobiyama *et al.* (2006)	Material de referência palestra webnar - Monitoramento Hidrológico na Gestão e Recursos Hídricos Ferramentas, Avanços e Desafios.	Todos

Fonte: Autor.

4.2 Área de monitoramento e versionamento da API REST.

A Figura 33 apresenta o mapa ilustrando a localização das PCDs administradas pela parceria entre a Agência para o Desenvolvimento da Bacia da Lagoa Mirim e o grupo de pesquisas HIDROSEDI-UFPEL, utilizadas durante processo de desenvolvimento e testes da primeira versão da API REST. Foram registrados dados de no qual a PCDs realiza uma requisição POST ao servidor através de uma *url*, que após conferências, é registrada em banco de dados. A primeira versão da API REST não possui nenhuma AS ou DS, sendo implementada sem nenhuma metodologia vigente (Figura 11).

Figura 33 – Localização das PCDs distribuídas na Bacia Hidrográfica Mirim-São Gonçalo.



Fonte: Núcleo de Pesquisa Ensino e Extensão em Hidrometria e Sedimentos para o manejo de Bacias Hidrográficas - NEPE-HidroSedi..

A segunda versão da API REST foi implementada utilizando a metodologia apresentada na presente dissertação. Suas funcionalidades foram separadas de acordo com as responsabilidades resultantes da modelagem do domínio (Figura 31). Houve uma redução significativa de linhas de código utilizadas nas *Views* responsáveis pela operação *POST* realizada pela PCDs. As demais funcionalidades de paginação, filtro, *caching*, foram implementadas em como elementos Utilitários, desta forma podem ser importados e utilizados em qualquer lugar da aplicação. As funcionalidades de estatística descritivas das séries retornadas também foram implementadas como utilitários e posteriormente importadas para utilização em Casos de Uso.

4.3 Estrutura JSON e avaliação de desempenho

O principal objetivo de uma API REST é a obtenção de um pacote de dados em formato JSON, previamente preparados e estruturados para serem utilizados em uma SW *front-end*. Tal obtenção deve obedecer a critérios que garantam a boa experiência do usuário, sendo a velocidade de retorno dos dados a principal delas, ou seja, a API REST deve oferecer um desempenho satisfatório ao fornecer seus recursos.

Para a realização dos testes, ambas versões das APIs foram instaladas localmente. Para a eliminação de qualquer tipo de dúvida em relação aos resultados, foi utilizado um algoritomo para a realização de requisições *POST* aleatórias através da ferramenta *cURL.* Foram criados dados aleatórios com valores de variáveis hidrológicas: precipitação (mm), temperatura (°C), umidade relativa (%), velocidade do vento (km/h), radiação solar (w/m²). O período de criação corresponde a 01/01/2024 a 31/12/2024, com intervalo de tempo de 15 minutos, totalizando 175.680 valores em banco de dados *SQL* (*Structured Query Language*).

Foram compostas duas estruturas em formato *JSON,* uma estrutura aninhada e uma estrutura não aninhada (Figura 34). Quatro grupos de retorno de dados paginados, sendo eles 50, 100, 200 e 400, e quatro cenários de testes:

- Cenário 1 – API REST primeira versão e dados não aninhados.
- Cenário 2 – API REST segunda versão e dados não aninhados.
- Cenário 3 – API REST versão e dados aninhados.
- Cenário 4 – API REST segunda versão e dados aninhados.

As requisições *GET* foram realizadas com o auxílio da ferramenta POSTMan. Para cada cenário foram realizadas três requisições e adotado a média como resultado. Para verificar possível influência da ORM do *Django,* também foram realizadas consultas com *SQL,* diretamente no banco de dados

.

Figura 34 - Estruturas de retorno em formato JSON, aninhada e não aninhada.

```
{
  "usuario": {
    "id": 1,
    "email": "usuario@example.com",
    "is_staff": false,
    "is_active": true,
    "date_joined": "2024-02-12T14:30:00Z",
    "plataformas_coleta_dados": [
      {
        "id": 101,
        "nome": "Plataforma Norte",
        "localidade": "Cidade A",
        "sensores": [
          {
            "id": 201,
            "codigo": "SENSOR-001",
            "nome": "Sensor de Temperatura",
            "unidade": "°C"
          }
        ],
        "leituras": [
          {
            "id": 301,
            "tempo_registro": "2024-02-12T15:00:00Z",
            "leituras_sensores": [
              {
                "id": 401,
                "valor_dado": 25.3,
                "sensor": {
                  "id": 201,
                  "codigo": "SENSOR-001",
                  "nome": "Sensor de Temperatura",
                  "unidade": "°C"
                }
              }
            ]
          }
        ]
      }
    ]
  }
}
```

Estrutura JSON - Aninhada

```
{
  "usuario": {
    "id": 1,
    "email": "usuario@example.com",
    "is_staff": false,
    "is_active": true,
    "date_joined": "2024-02-12T14:30:00Z"
  },
  "plataforma_coleta_dados": {
    "id": 101,
    "nome": "Plataforma Norte",
    "localidade": "Cidade A",
    "usuario_id": 1
  },
  "pcd_sensor": {
    "id": 201,
    "codigo": "SENSOR-001",
    "nome": "Sensor de Temperatura",
    "unidade": "°C",
    "plataforma_coleta_dados_id": 101
  },
  "pcd_leitura": {
    "id": 301,
    "tempo_registro": "2024-02-12T15:00:00Z",
    "plataforma_coleta_dados_id": 101
  },
  "pcd_leitura_sensor": {
    "id": 401,
    "valor_dado": 25.3,
    "pcd_leitura_id": 301,
    "pcd_sensor_id": 201
  }
}
```

Estrutura JSON – Não Aninhada

Fonte: Autor.

Os Cenários 1 e 2 não apresentaram diferenças de desempenho significativas, em requisições realizadas pela ORM do *Django,* por outro lado nota-se um aumento no tempo de resposta em requisição *SQL* realizada com 400 dados, na segunda versão da API REST, tendo um aumento no tempo da resposta da requisição de 118, 75 %. A Tabela 4 apresenta os resultados do teste de desempenho realizado no Cenário 1

Tabela 4 – Resultados teste de desempenho do Cenário 1.

Cenário 1 - API REST Primeira Versão - Dados Não Aninhados			
N° pacotes	Tamanho dados (MB)	Tempo de Resposta (ms)	Tempo de Resposta SQL (ms)
50	0,6	0,04	0,03
100	1,2	0,06	0,02
200	2,4	0,16	0,03
400	4,8	0,66	0,16

Fonte: Autor.

A Tabela 5 apresenta os resultados do teste de desempenho realizado no Cenário 2.

Tabela 5 – Resultados teste de desempenho do Cenário 2.

Cenário 2 - API REST Segunda Versão - Dados Não Aninhados			
N° pacotes	Tamanho dados (MB)	Tempo de Resposta (ms)	Tempo de Resposta SQL (ms)
50	0,6	0,03	0,04
100	1,2	0,09	0,03
200	2,4	0,19	0,13
400	4,8	0,78	0, 35

Fonte: Autor.

Os Cenários 3 e 4 também não apresentaram diferenças significativas nas requisições realizadas pela ORM do *Django* em 50 e 100 dados paginados, porém em 200 e 400 o Cenários três excedeu o tempo de requisição, tendo retornado código de status 500, o que indica erro no servidor. O Cenário 4 apresentou resultados de retorno com 200 e 400 dados paginados, porem com tempos insatisfatórios para uma boa experiência de usuário.

Tabela 6 - Resultados teste de desempenho do Cenário 3.

Cenário 3 - API REST Primeira Versão - Dados Aninhados			
N° pacotes	Tamanho dados (MB)	Tempo de Resposta (ms)	Tempo de Resposta SQL (ms)
50	0,6	0,48	0,04
100	1,2	1,06	0,12
200	2,4	Falha	0, 23
400	4,8	Falha	0,22

Fonte: Autor

Tabela 7 - Resultados teste de desempenho do Cenário 4.

Cenário 4 - API REST Segunda Versão - Dados Aninhados			
N° pacotes	Tamanho dados (MB)	Tempo de Resposta (ms)	Tempo de Resposta SQL (ms)
50	0,6	0,39	0,04
100	1,2	0,96	0,10
200	2,4	8.341	0,016
400	4,8	46.731	0,019

Fonte: Autor.

Com exceção dos resultados dos cenários 3 e 4, os resultados dos testes não apresentam um padrão ou diferenças, não sendo possível determinar se a implementação arquitetura exerceu influência nos resultados. As avaliações foram realizadas em uma máquina com memória RAM de 4GB, processador Intel® Core™ i3 – 5005U CPU 2.000GHz, sendo relativamente antiga e com pouco poder processamento, esse fator também pode ter afetado na aleatoriedade dos resultados.

No contexto isolado de cada cenário, as requisições realizadas pela ORM *Django* e *SQL* apresentaram diferenças notórias em seu tempo de retorno, em ambas estruturas e quantidades de dados paginados. Porém, a elaboração da requisição *SQL,* para a estrutura aninhada apresenta complexidade elevada, o que confronta o objetivo de utilização *frameworks*, que é a simplificação da implementação de um SW.

Considerando as estruturas JSON apresentadas, de modo geral a estrutura não aninhada apresentou melhores resultados nos tempos de retorno em todos números de dados paginados, porem a desvantagem em relação aos dados aninhados se da na necessidade de maior manipulação, dependendo do contexto de utilização desejado.

5 Considerações finais

A utilização da arquitetura REST para a construção de APIs abre várias possibilidades de empregabilidade em diversos setores que necessitam de monitoramento de dados em tempo real, em destaque as diversas áreas da engenharia. Durante a pesquisa para a construção da base bibliográfica deste trabalho foram constatadas diversas publicações de aplicações na área de engenharia

agrícola e/ou agronomia, devido ao avanço e possibilidade de investimento para a implementação da Agricultura 4.0.

Órgãos do Ministério da Agricultura, Pecuária e Abastecimento, em conjunto com iniciativa privada, vem investindo na capacitação de profissionais de engenharia e áreas de tecnologia da informação com o objetivo de diminuir a lacuna da falta de conhecimento entre profissionais sobre respectivas áreas, desta forma possibilitando a criação de tecnologias a serem empregadas com objetivo de maximização e melhor controle de produções agrícolas (Brasil, 2021) .

Por outro lado, se tratando da area de engenharia hídrica e gestão de recursos hídricos, os trabalhos encontrados foram de número reduzido, sendo em sua maioria focados na comunicação de sistemas terceiros já consolidados. O número se atenua ao se tratar da criação de sistemas de alerta para o risco de desastres naturais de cunho hidrológico, e aplicações em pequenos municípios. Desta forma, espera-se que o presente trabalho possa exercer o papel de diretriz inicial para profissionais da engenharia hídrica, gestores de águas e tecnologia da informação na concepção e implementação de sistemas de combate e mitigação nas consequências da ocorrência de desastres naturais.

Foram apresentados os conceitos de AS e DS com argumentos a respeito dos fatores positivos de sua utilização, arquitetura REST e suas ferramentas tecnológicas de auxílio na implementação de código. A metodologia de modelagem orientada a domínio foi explicitada, de acordo com o contexto do *Django* e *Django REST framework.* Os itens 2.3 e 2.4 apresentaram a teoria para a definição do domínio durante o processo de modelagem, entretanto, a modelagem orientada a domínio pode ser considerada um processo de "*Overengineering*", ou seja, complexidade desnecessária, quando utilizados em *frameworks web*. A comunidade de desenvolvedores de estabeleceram a seguinte pergunta " A proposta da API REST se trata de um simples CRUD ?", se a resposta for sim. Não utilize metodologias de AS e DS, caso a resposta seja negativa, cabe ao desenvolvedor avaliar a utilização.

Como continuidade deste trabalho, sugiro a implementação de um SW com foco na interação com o usuário (*front-end*). Considerando o domínio apresentado, a funcionalidade inicial de apresentação das estatísticas descritivas da precipitação e nível d'água considerando o limiar de inundação. Ao se utilizar as técnicas de modelagem orientada ao domínio apresentadas neste trabalho, é possível a

implementação modular de cada aplicação DRF, separando as responsabilidades e facilitando futuras manutenções e ampliação do SGD.

.

Referências

ALVES, Maria Eduarda. **Manual de aplicação do modelo MGB utilizando IPH-Hydro Tools**: manual técnico. Porto Alegre: Hge, Iph, Ufrgs, 2020. 5 v.

ANA (org.). **Especificações técnicas**: plataforma de coleta de dados (pcds). Brasília: Agência Nacional das Águas, 2011.

BRASIL. **Decreto N° 10.593**. Brasil, 24 dez. 2020. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2019-2022/2020/decreto/d10593.htm. Acesso em: 19 set. 2024.

BRASIL. **Lei N° 12.608**. Brasil, 10 dez. 2012. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2011-2014/2012/lei/l12608.htm. Acesso em: 19 set. 2024.

BRASIL. **Lei N° 9.433**. Brasil, 08 jan. 1997. Disponível em: https://www.planalto.gov.br/ccivil_03/leis/l9433.htm. Acesso em: 19 set. 2024.

BRASIL. MINISTÉRIO DA AGRICULTURA PECUÁRIA E DESENVOLVIMENTO. . **Potencialidades e Desafios do Agro 4.0**. Brasilia: Mapa/Aces, 2021.

BROCK, Rebeca Wirfs; MCKEAN, Alan. **Object Design**: roles, responsabilities, and collaborations. [S.L]: Adison Wesley, 2002.

CEMADEN: Centro Nacional de Monitoramento e Alerta de Desastres Naturais. Centro Nacional de Monitoramento e Alerta de Desastres Naturais. 2024. Disponível em: https://www.gov.br/cemaden/pt-br. Acesso em: 28 nov. 2024.

CHOW, Ven Te; MAIDMENT, Davir R.; MAYS, Larry W.. **Applied Hydrology**. [S. L.]: McGraw-Hill, 1988.

CLARKE, Robin T.; DIAS, Pedro L. da Silva (org.). **As necessidades de observação e monitoramento dos ambientes brasileiros quanto aos recursos hídricos**. Brasília: Ct-Hidro, 2003.

COLLISWHONN, Walter; DORNELLES, Fernando. **Hidrologia para a Engenharia e Ciências Ambientais**. 2. ed. Porto Alegre: Abrh, 2015.

EVANS, Eric. **Domain-Driven Design**: tackling complexity in the heart of the software. [S.L]: Adison Wesley, 2004
.
FARIAS, Josiane Rosa Costa *et al*. Modelo para grandes bacias e suas aplicações: uma revisão bibliográfica. In: SIMPOSIO BRASILERO DE RECURSOS HPIDRICOS, 25., 2023, Sergipe. **Anais [...] .** Sergipe: Abrhidro, 2023. p. 1-10.

FERREIRA, Glaucia Machado *et al*. Utiização de modelo digital de elevação hidrologicamente consistente na obtenção de características. In: SIMPOSIO BRASILEIRO DE RECURSOS HÍDRICOS, 19., 19, Maceió. **Anais [...] .** Maceió: Abrhidro, 2011. p. 1-20.

FIELDING, Roy Thomas. **Architectural Styles and The design of Network-based Software Aerchitectures**. 2000. 180 f. Dissertação (Mestrado) - Curso de Doctor Of Philosophy In Information And Computer SCience, University Of California, Irvine, 2000. Disponível em: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 08 out. 2024.

FIORIN, Tatiana Tasquetto; ROSS, Meridiana dal. **Climatologia Agrícola**. Santa Maria: Colégio Politécnico Ufsm, 2015.

GAMMA, Erich *et al*. **Design Patterns**: elements of reusable object-oriented software. [S.L]: Addison Wesley, 1994.

GITHUB (org.). **State of the Octoverse**. Disponível em: https://github.blog/news-insights/octoverse/. Acesso em: 29 dez. 2024.

GOMES, Isabelle Sena; CAMINHA, Iraquitan de Oliveira. Guia para estudos de revisão sistemática: uma opção metodológica para as ciências do movimento humano. **Movimento**, Porto Alegre, v. 20, n. 01, p. 395-411, Não é um mês valido! 2014. Trimestral.

GUANDIQUE, Manuel Enrique Gamero; MORAIS, Leandro Cardoso de. Estudo de váriáveis hidrológicas e do balanço hídrico em bacias hidrográficas. In: POMPEO, Marcelo *et al*. **Ecologia de reservatórios e interfaces**. São Paulo: Instituto de Biociências -Ib/Usp, 2015. p. 434-447.

HILLAR, Gaston C.. **Django RestFull Web Services**: the easiest way to build python restful apis web services with django. Berminghan: Packt, 2018.

HORNBERGER, George M.. **Elements of Physical Hydrology**. 2. ed. Baltmore: Johns Hopkins University Press, 2014.

KALINOWSKI, Marcos. **Engenharia de Software para Ciência de Dados**: um guia básico em ênfase na construção de sistemas machine learning com python. São Paulo: Casa do Código, 2023.

KEMMERICH, Pedro Camargo. **Desenvolvimento de um sistema de telemetria via GPRS de baixo custo para uma estação de monitorament**. 2019. 126 f. TCC

(Graduação) - Curso de Engenharia de Controle de Automação, Universidade Federal de Santa Maria, Santa Maria, 2019.

KOBIYAMA, Masato *et al*. **Aprender Hidrologia para Prevenção de Desastres Naturais**. Florianópolis: UfSC - Departamento de Engenharia Ambiental, 2009.
KOBIYAMA, Masato *et al*. **Prevenção de Desastres Naturais**: conceitos básicos. Florianópolis: Organic Trading, 2006.

LIMA, Marciano da Costa; FREIMAN, Fabiano Peixoto; CAMBOIM, Silvana Philippi. Desenvolvimento de uma aplicação web de mapeamento colaborativo para identificação de áreas de risco. **Revista de Geociências do Nordeste**, [S.L.], v. 8, n. 1, p. 217-226, 14 jun. 2022. Universidade Federal do Rio Grande do Norte - UFRN. http://dx.doi.org/10.21680/2447-3359.2022v8n1id26542.

LOTT, Steven F.; PHILIPS, Dusty. **Python Object-Oriented Programming**: build robust and maintainable object-oriented python applications and libraries. 4. ed. Birmingham: Packet Publishing Ltd., 2021.

MANTOVANI, Alice de Fátima da Fonseca. **Estudo Teórico da Arquitetura de Software Model View Controller**. 2021. 63 f. TCC (Graduação) - Curso de Sistemas de Informação, Faculdade de Tecnologia, Universidade Estadual de Campinas, Limeira, 2021.

MARTIN, Robert C.; GRENNING, James; BROWN, Simon. **Arquitetura Limpa**: o guia do artesão para estrutura e design de software. Rio de Janeiro: Alta Books, 2019.

MELÉ, Antonio. **Django 5 by exemple**: build porwerful an reliable python web applications from scratch. 5. ed. [S.L]: Packt, 2024.

MOREIRA, Bruno Medeiros. **Desenvolvimento de aplicativo web para a visualização de informação hidrológica**. 2024. 124 f. Dissertação (Mestrado) - Curso de Engenharia Informática e Técnologia Web, Universidade Aberta, Lisboa, 2024.

MOZILLA. **Mdn-Web-Docs**: resources for developers by developers. Resources for developers by developers. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview. Acesso em: 02 nov. 2024.

NAGHETTINI, Mauro. **Introdução á Hidrologia Aplicada**. Belo Horizonte: Eeufmg, 2012.

NAGHETTINI, Mauro; PINTO, Éber José de Andrade. **Hidrologia Estatística**. Belo Horizonte: Cprm, 2007.

PERCIVAL, Harry J. W.; GREGORY, Bob. **Architecture patterns with python**: enable test-driven development, domain drive design, and event-driven microservices. [S.L]: O'Reilly, 2020.

PEREIRA, Marco Alésio Figueiredo; BARBIEIRO, Bruno Lippo; QUEVEDO, Daniela Muller de. Importância do monitoramento e disponibilização de dados hidrológicos para a gestão integrada dos re. **Sociedade & Natureza**, [S.L.], v. 32, p. 308-320, 6 maio 2020. EDUFU - Editora da Universidade Federal de Uberlandia. http://dx.doi.org/10.14393/sn-v32-2020-43458.

PRESSMAN, Roger S.; MAXIM, Brucer R.. **Engenharia de Software**: uma abordagem profissional. 9. ed. Porto Alegre: Amgh, 202

REENSKUG, Trygve. **The original MVC Reports**. Elabora por Dept. of informatics University of Oslo. Disponível em: https://folk.universitetetioslo.no/trygver/2007/MVC_Originals.pdf. Acesso em: 30 dez. 2024.

RICARDO, Ana M. *et al*. Flood simulation with the RiverCure approach: the open dataset of the 2016 águeda flood event. **Earth System Science Data**, [S.L.], v. 16, n. 1, p. 375-385, 16 jan. 2024. Copernicus GmbH. http://dx.doi.org/10.5194/essd-16-375-2024.

ROBAINA, Luis Eduardo de Souza. **Desastres Hidrologicos**: levantamento para o estado do rio grande do sul, brasil. Canoas: Mérida, 2024.

ROCHA, José Gladistone. Arquitetura em camadas com o uso do paradigma MVC e processo unificado na programação de software or. **Tecnologia em Projeção**, [S.L], v. 9, n. 1, p. 31-49, 28 set. 2018. Semestral. Disponível em: https://projecaociencia.com.br/index.php/Projecao4/issue/view/96. Acesso em: 24 out. 2024.

SANTOS, Irani dos. **Hidrometria Aplicada**. Curitiba: Lactec, 2001.

SANTOS, Keyla Almeida dos; MATOS, Artur José Soraes (org.). **Relatorio de Atividades**: levantamentos, estudos, previsão e alerta de eventos críticos.. [S.L]: Cprm, 2021.

STAIR, Ralph M.; REYNOLDS, George W.. **Princípios de sistemas de informação**. 11. ed. São Paulo: Cengage, 2016.

THAINES, Pedro; MORALES, Analucia Shiaffino. **Sistema integrado com telemetria e mensageria para auxiliar o manejo de irrigação agrícola através d**. 2022. 24 f. TCC (Graduação) - Curso de Engenharia de Computação, Centro de Ciências, Tecnologias e Saúde, Universidade Federal de Santa Catarina, Araranguá, 2022.

UNESP, Faculdade de Ciências Agrônomicas (org.). **Tipos de Revisão de Literatura**. Botucatu: Biblioteca Prof. Paulo de Carvalho Mattos, 2015.

VICENT, William S.. **Django for APIs**: build web apis with django and python. 2. ed. [S.L]: Leanpub, 2022.

WILSON, Greg. **Software Design by exemple**: a tool-based introduction with python. [S.L]: Crc Press, 2024.

YNOUE, Rita Yuri. **Metereologia**: noções basicas. São Paulo: Oficina de Textos, 2017.

ZENG, Xuan; TU, Zhen Yu; MA, Yong Li. Research on the Hydrology Telemetry System of Poyang Lake Based on GPRS. **Advanced Materials Research**, [S.L.], v. 311-313, p. 1319-1322, ago. 2011. Trans Tech Publications, Ltd.. http://dx.doi.org/10.4028/www.SWientific.net/amr.311-313.1319.