**UNIVERSIDADE FEDERAL DE SANTA MARIA**

**CENTRO DE TECNOLOGIA**

**CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO**

**INTERFACE DE SUPERVISÓRIO PARA SISTEMAS DE RASTREAMENTO SOLAR - TRACKER**

**BRUNO GABRIEL FLORES SAMPAIO**

**Santa Maria, RS, Brasil**

**2023**

**RESUMO**

Este projeto visa desenvolver um sistema supervisório de controle para um rastreador solar do tipo Tracker, que utiliza o protocolo de comunicação Modbus e algoritmos de rastreamento solar SPA (Sun Position Algorithm), implementados em Python com o auxílio do framework de design de aplicativos chamado Kivy. O rastreador solar Tracker é um dispositivo que acompanha o movimento do sol ao longo do dia para maximizar a eficiência na captação de energia solar. O sistema supervisório de controle permite monitorar e controlar o rastreador solar de forma centralizada, fornecendo informações sobre a posição do sol e permitindo ajustes precisos para otimização do seu desempenho. Para a comunicação entre o sistema supervisório e o rastreador solar, o protocolo Modbus é adotado devido ao fato dele ser amplamente utilizado em aplicações industriais, oferecendo uma estrutura simples e confiável para a troca de dados entre dispositivos. Ele permite a leitura e escrita de informações, como a posição atual do rastreador solar, velocidade de rastreamento, dados de sensores e outros parâmetros relevantes. Além disso, são utilizados algoritmos de rastreamento solar SPA, que são responsáveis por calcular a posição do sol em relação à localização geográfica do rastreador solar em tempo real. Esses algoritmos consideram informações como a latitude, longitude e horário atual para determinar a posição precisa do sol no céu. Essas informações são utilizadas para controlar os movimentos do rastreador solar e garantir o direcionamento ideal em direção à fonte de luz. O desenvolvimento do sistema é realizado em Python, uma linguagem de programação versátil e popular, escolhida pela sua facilidade de uso, vasta biblioteca de suporte e excelente integração com o protocolo Modbus. O framework Kivy é utilizado para criar a interface de usuário do sistema, proporcionando uma experiência visual agradável e interativa. Esse sistema contribui para a eficiência na captação de energia solar e o aproveitamento máximo dos recursos disponíveis. O sistema possui um login eficaz, com um banco de dados rodando no *backend*, que armazena as credenciais de *login*, que pode ser utilizado local e remotamente, além disso possui uma interface fluida, pois esta baseado em programação concorrente, utiliza ndo *Threads.* Para a comunicação com o sistema físico utilizado, a velocidade nas trocas de mensagens via modbus para aquisição da posição dos motores e medição de energia dos painéis foram efetuadas em tempos médios de 11ms.

**LISTA DE FIGURAS**

[Figura 1 : Mapa do potencial de geração solar fotovoltaica em termos do rendimento energético anual para todo o Brasil (medido em kWh/kWp.ano ) 10](#_Toc14824)

[Figura 2 : Pirâmide de um sistema SCADA 14](#_Toc31585)

[Figura 3 : Exemplo de MODEM de comunicação. 15](#_Toc15245)

[Figura 4 : Logo da linguagem de programação Python. 19](#_Toc24019)

[Figura 5 : Primeiro CLP da Modcon - MODICON 084 21](#_Toc856)

[Figura 6 : Logo da ModBus Organization. 21](#_Toc11530)

[Figura 7 : Modelo de separação dos campos de dados no protocolo Modbus. 22](#_Toc3861)

[Figura 8 : Pacotes](#_Toc10462) *[Modbus](#_Toc10462)* [RTU e TCP/IP. 23](#_Toc10462)

[Figura 9 : Modelo mestre-escravo](#_Toc3577) *[modbus](#_Toc3577)*[. 23](#_Toc3577)

[Figura 10 : Modelo de requisição e resposta Modbus. 24](#_Toc28598)

[Figura 11 : Modelos de PDUs do modelo](#_Toc13373) *[Modbus](#_Toc13373)*[. 25](#_Toc13373)

[Figura 12 : Registradores Modbus. 25](#_Toc3915)

[Figura 13 : Modelo de hierarquia de objetos em interfaces orientadas a objetos. 28](#_Toc29196)

[Figura 14 :](#_Toc4783) *[Display](#_Toc4783)* [utilizado como IHM. 31](#_Toc4783)

[Figura 15 : Hierarquia do modelo de organização MVC 33](#_Toc4511)

[Figura 16 : Modelo de requisição de](#_Toc12685) *[login](#_Toc12685)*[. 34](#_Toc12685)

[Figura 17 : Fluxograma do processo de controle do nível de acesso 35](#_Toc167)

[Figura 18 : Fluxograma das responsabilidades do Modelo. 37](#_Toc1202)

[Figura 19 : Fluxograma relacional entre o Modelo e o gerenciador do servidor remoto. 38](#_Toc18667)

[Figura 20 : Fluxograma entre Modelo e interface serial RS-485. 39](#_Toc25548)

[Figura 21 : Tela de](#_Toc31794) *[login](#_Toc31794)*[. 41](#_Toc31794)

[Figura 22 : Acesso ao menu de login. 42](#_Toc20594)

[Figura 23 : Seção para login. 42](#_Toc5987)

[Figura 24 : Tela de registro de novo usuário. 43](#_Toc4709)

[Figura 25 : Mensagem de erro de autenticação. 44](#_Toc8427)

[Figura 26 : Visualização da tela inicial. 45](#_Toc26902)

[Figura 27 : Descrição dos objetos presentes na tela inicial. 45](#_Toc1631)

[Figura 28 : Visualização da tela de mapas do sistema. 46](#_Toc30312)

[Figura 29 : Visualização detalhada da tela de mapas do sistema 46](#_Toc32098)

[Figura 30 : Visualização da tela de conexão serial. 47](#_Toc18102)

[Figura 31 : Visualização detalhada da tela de conexão serial. 47](#_Toc27594)

[Figura 32 : Visão geral do Raspberry Pi Pico 48](#_Toc6680)

[Figura 33 : Motor DC escovado utilizado no sistema. 49](#_Toc6910)

[Figura 34 : Ponte-H L298N de dois canais. 50](#_Toc21218)

[Figura 35 : Sensor AS5600 (a) Visão geral do sensor (b) Imã com polaridade diametral 50](#_Toc15010)

[Figura 36 : Visão geral da placa com RTC embarcado. 51](#_Toc11630)

[Figura 37 : Relé de acionamento 12V/24V@10A RAS-1210 51](#_Toc30327)

[Figura 38 : Esquemático de acionamento do relé RAS-1210 52](#_Toc9500)

[Figura 39 : Configuração do FT232R para conversor USB para UART 53](#_Toc9084)

[Figura 40 : Módulo funduino utilizando o FT232R como conversor. 53](#_Toc10902)

[Figura 41 : Foto do sistema completo. 54](#_Toc18522)

[Figura 42 : Troca de mensagens TCP/IP entre sistema e servidor. 56](#_Toc3929)

[Figura 43 : Informações trafegadas para criar novo usuário (a) Lado supervisório (b) Lado servidor. 57](#_Toc1278)

[Figura 44 : Mensagem para criação de novo usuário. 57](#_Toc19057)

[Figura 45 : Registro de usuário no banco de dados do servidor. 58](#_Toc7560)

[Figura 46 : Captura dos pacotes transitados pela USB via](#_Toc31363) *[Wireshark USBCap](#_Toc31363)*[. 59](#_Toc31363)

[Figura 47 : Conteúdo do frame USB. 59](#_Toc11364)

[Figura 48 : Pacote de dados recebido do sistema físico como resposta. 60](#_Toc18705)

[Figura 49 : Medição do tempo de resposta do dispositivo Modbus. 61](#_Toc16310)

[Figura 50 : Mensagens trocadas para a leitura de todos registradores Modbus do sistema. 61](#_Toc4858)

[Figura 51 : Visualização da funcionalidade dos gráficos de posição do sistema. 62](#_Toc26108)

**SUMÁRIO**

[1. Introdução 6](#_Toc5093)

[1.1 Aspectos gerais 6](#_Toc2089)

[1.2 Antecedente do problema 9](#_Toc32256)

[1.3 Descrição do problema 11](#_Toc711)

[1.4 Objetivo 11](#_Toc26369)

[1.5 Objetivos específicos 11](#_Toc12881)

[2. revisão bibliográfica 13](#_Toc14676)

[2.1 O que é um sistema SCADA 13](#_Toc13991)

[2.2 Protocolos de comunicação 15](#_Toc24588)

[2.3 Aplicações 17](#_Toc13939)

[3. MATERIAIS E MÉTODOS 19](#_Toc22378)

[3.1 Especificações de](#_Toc29743) *[Software](#_Toc29743)* [- Python 3.10 19](#_Toc29743)

*[3.2](#_Toc6742)* [Comunicação de dados –](#_Toc6742) *[Modbus](#_Toc6742)* [20](#_Toc6742)

[3.2.1 Estrutura dos pacotes](#_Toc42) *[Modbus](#_Toc42)* [22](#_Toc42)

[3.2.2 Modelo de comunicação mestre-escravo 23](#_Toc7364)

[3.2.3 Código de função 24](#_Toc6118)

[3.2.4 Tipos de registradores 25](#_Toc16927)

[3.3 Interface gráfica -](#_Toc1356) *[Kivy](#_Toc1356)* [26](#_Toc1356)

[3.4 Banco de dados -](#_Toc30764) *[SQLite3](#_Toc30764)* [28](#_Toc30764)

[3.5 Decisão 29](#_Toc9345)

[3.6 Especificações de Hardware 30](#_Toc8721)

[3.6.1 Raspberry Pi 3 30](#_Toc11620)

[3.6.2 IHM 30](#_Toc7251)

[4. DESENVOLVIMENTO 32](#_Toc1533)

[4.1 Software de controle 32](#_Toc10766)

[4.1.1 Organização do Software de interface 32](#_Toc19811)

[4.1.2 Processos internos 33](#_Toc23737)

[4.1.3 Modelo 36](#_Toc2794)

[4.1.4 Tela gráfica 40](#_Toc21292)

[5. INTERFACE DE CONTROLE 41](#_Toc2962)

[5.1 Tela de Login 41](#_Toc20940)

[5.2 Tela de inicio 44](#_Toc2058)

[5.3 Tela de mapas 46](#_Toc16331)

[5.4 Tela de conexão serial 47](#_Toc2132)

[6. SISTEMA físico 48](#_Toc23066)

[6.1 Microcontrolador Raspberry Pi Pico 48](#_Toc24598)

[6.2 Motores de corrente continua (DC) 49](#_Toc9193)

[6.3 Driver de acionamento de motores DC 49](#_Toc16398)

[6.4 Sensores de posição angular magnéticos 50](#_Toc14832)

[6.5 Relógio de tempo real 51](#_Toc4551)

[6.6 Dispositivos de desarme de alimentação 51](#_Toc26749)

[6.7 Interface USB-UART 52](#_Toc32151)

[6.8 Fonte de alimentação 53](#_Toc5830)

[6.9 Sistema completo 54](#_Toc5424)

[7. RESULTADOS 55](#_Toc451)

[7.1 Autenticação de usuário 55](#_Toc29580)

[7.1.1 Registro e usuários 55](#_Toc27350)

[7.1.2 Autenticação de login 58](#_Toc1397)

[7.2 Comunicação com o sistema 58](#_Toc16596)

[Considerações finais 63](#_Toc1776)

[REFERÊNCIAS 66](#_Toc15911)

[ANEXO 68](#_Toc25936)

[APÊNDICE 69](#_Toc3356)

# Introdução

## Aspectos gerais

Precedendo a era heliocentrista, baseados nos fortes argumentos propostos por Aristóteles (384 a.C. a 322 a.C.) e Ptolomeu (90 d.C. – 168 d.C.) de que a Terra era o centro do universo, adotou-se a visão geocentrista do mundo. Não é descabível pensar na terra ocupando o centro do universo e todos os astros girando em seu torno quando leis fundamentais da natureza ainda eram desconhecidas e as teorias eram obtidas com bases experimentais e através da pura observação, somados a recursos muito limitados.

No entanto, rompendo tais ideologias e indo contra a verdade adotada na época, Nicolau Copérnico (1473-1543), no ano de 1543, já em seu leito de morte, publicou uma obra “Da revolução de esferas celestes”, tradução livre para o português de *De Revolutionibus Orbium Coelestium*, livro que continha todos os anos de pesquisa de Copérnico acerca do movimento do sol e os demais corpos celestes conhecidos na época com forte embasamento matemático. Além disso, Copérnico pela primeira vez apresentou publicamente a ideia de que o planeta Terra realiza 3 tipos de movimentos, sendo eles: O movimento de rotação em torno do seu próprio eixo (Rotação diária), o movimento de translação, que realiza ao redor do sol (Volta anual) e o movimento de precessão que realiza em torno do seu eixo eclíptico (inclinação anual de seu eixo). Tal livro gerou uma verdadeira revolução na maneira como o mundo era visto na época e foi o responsável por posteriormente derrubar por terra as teorias Geocentristas vigentes na época (COPÉRNICO, 2003).

Servindo de inspiração para outros grandes nomes da física, como Johannes Kepler (1571 - 1630) que em seu livro “Astronomia Nova” publicado em 1610, apresentava as suas duas primeiras leis acerca do movimento dos planetas ao redor do sol (TOSSATO; MARICONDA, 2010), confrontando as teorias heliocentristas e trazendo a atenção dos cientistas para esse astro.

Posteriormente, Galileu Galilei (1564 – 1642) com seu folheto “Mensageiro Sideral”, tradução livre de “*Sidereus Nuncius*”, publicado em 1610, apresentou observações feitas do espaço através de telescópios desenvolvidos pelo próprio Galileu, trazendo novos fatos acerca do que viria ser descoberto como a Via Láctea e o universo, trazendo por terra as ideias geocentristas.

Esses cientistas foram muito importantes nos seus campos de pesquisa e dedicaram suas vidas, arriscando-se para fazer ciência em um tempo que isso não era permitido, pois iniciaram uma busca por conhecimento nos céus, estudando os astros que nos rodeiam e trazendo informações a cerca deles, que formam a base de tudo que se é transmitido de conhecimento e aceito hoje.

Com as descobertas do comportamento dos astros, pode-se então compreender suas características de movimento, traçar trajetórias, calcular suas posições e fazer previsões de posição de cada um no céu de maneira precisa. Quando pega-se o Sol como astro de estudo, tais feitos se tornam ainda mais importantes uma vez que o sol é uma grande fonte de energia, principalmente nos dias de hoje, do qual é possível aproveitar sua energia, não apenas no âmbito da agricultura ou arquitetura, mas também como uma fonte geradora de energia elétrica inesgotável.

Com as descobertas de Alexandre Edmond Becquerel em 1839, quando descobriu o efeito fotovoltaico, efeito que transforma a energia dos raios solares em energia elétrica, e Willoughby Smith que mais tarde, em 1873 descobriu a fotocondutividade do selênio, que originalmente era um isolante, mas se comportava como um condutor na presença de raios solares e não apenas conduzia eletricidade como também era capaz de gerá-la, teve-se em 1883 a criação da primeira célula fotovoltaica por Charles Fritts. E em 1958, Russell Ohl patenteou o primeiro sistema fotovoltaico, o mais próximo do que temos hoje (RICHARDSON, 2018), um sistema capaz de produzir energia elétrica através dos raios solares que poderia ser facilmente instalado em qualquer lugar.

Não se sabia na época que tais descobertas e invenções iriam mudar a forma como a geração de energia passou a ser realizada ao redor do mundo, tornando a geração solar uma das principais e mais cobiçadas fontes geradoras de energia renováveis no planeta. Atualmente há um grande número de incentivos em escala global para essa prática de geração. Um reflexo dos incentivos no panorama mundial, foi a produção primária de energia solar realizada por painéis fotovoltaicos, aumentando 395% nos períodos entre 2003 e 2013, frente ao aumento de 56% das outras fontes renováveis. Somente o crescimento da geração de energia eólica superou a energia solar nesse período (SILVA, 2015).

No Brasil, em 2021 a marca de geração solar está para atingir 8GW de potência instalada (sieBRASIL, sd), representando apenas 4% da geração de energia no país, como mostrado na Tabela 1, um valor baixo se comparado ao potencial de geração que esse tipo de tecnologia pode oferecer. Atualmente, a maior usina de geração solar do Brasil está localizada na cidade de São Gonçalo do Gurguéia, no Piauí. Contando com mais de 2,2 milhões de painéis solares em uma região semiárida do Brasil podendo chegar a gerar 2,2TW por ano de energia (GREEN POWER, 2021) sendo também considerada a maior usina de geração solar da América do Sul. A usina de São Gonçalo por sua vez, conta com um sistema de rastreio do sol com um grau de liberdade, sendo capaz de seguir o sol no seu movimento de azimute e zenite, variado sua inclinação com o solo devido sua posição geográfica favorecida.

Tabela 1: Relação dos diferentes tipos de geração elétrica no Brasil.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de geração** | **Unidade** | **Produção** | **Percentagem** |
| Hidrelétrica | MW | 109294,05 | 60,89% |
| Eólica | MW | 17146,13 | 9,55% |
| Solar | MW | 7922,22 | 4,41% |
| Térmica | MW | 43152,37 | 24,04% |
| Nuclear | MW | 1990,00 | 1,11% |
| Total | MW | 179504,77 | 100,00% |
| Valores referentes a 01 set 2021 | | | |

Fonte: adaptação de sieBRASIL, sd.

Ao contrário dos painéis móveis, os painéis solares fixos, que possuem uma limitação de irradiação que se agrava na medida em que se afasta das regiões próximas à linha do equador. Devido aos movimentos do sol ao longo do dia (rotação), somados aos movimentos do sol ao longo do ano (translação), os painéis fixos perdem grande parte da irradiação que poderiam receber no período do dia. Estima-se que 40% da energia solar é desperdiçada para configurações de painéis fixos em regiões afastadas da linha do equador quando comparados com painéis não fixos com 2 graus de liberdade e até 30% com apenas um grau de liberdade (VALLDOREIX GREENPOWER, 2015).

Visando um notável crescimento no uso de sistemas de rastreamento solar na geração de energia, esses sistemas têm se tornado uma escolha popular devido ao seu potencial de aumentar significativamente a eficiência e o rendimento das instalações solares. No entanto, à medida que as instalações solares com *Trackers* se tornam mais complexas e numerosas, surge a necessidade de um sistema supervisório confiável para gerenciar e controlar eficientemente esses sistemas, principalmente em redes complexas com numerosos sistemas *Tracker* pendurados na mesma rede.

Um sistema supervisório desempenha um papel crucial no controle e monitoramento desses sistemas, permitindo um posicionamento preciso dos painéis solares e a otimização do rendimento energético. O sistema supervisório coleta dados dos sensores, como a posição do sol, a intensidade da luz solar e a produção de energia, e os exibe de forma clara e intuitiva em uma interface gráfica. Com base nessas informações em tempo real, os operadores podem tomar decisões informadas para maximizar a eficiência da geração de energia solar.

Além disso, um sistema supervisório oferece recursos avançados de controle, permitindo que os operadores realizem ajustes e modifiquem as estratégias de movimento, incluindo a programação de perfis de movimento, a adaptação a condições climáticas variáveis ou o alinhamento com padrões específicos de geração de carga. Com a automação proporcionada pelo sistema supervisório, os ajustes e o controle dos sistemas de rastreamento podem ser realizados de forma eficiente e precisa, eliminando a necessidade de intervenção manual frequente e como consequência, reduzindo a necessidade de manutenções corretivas.

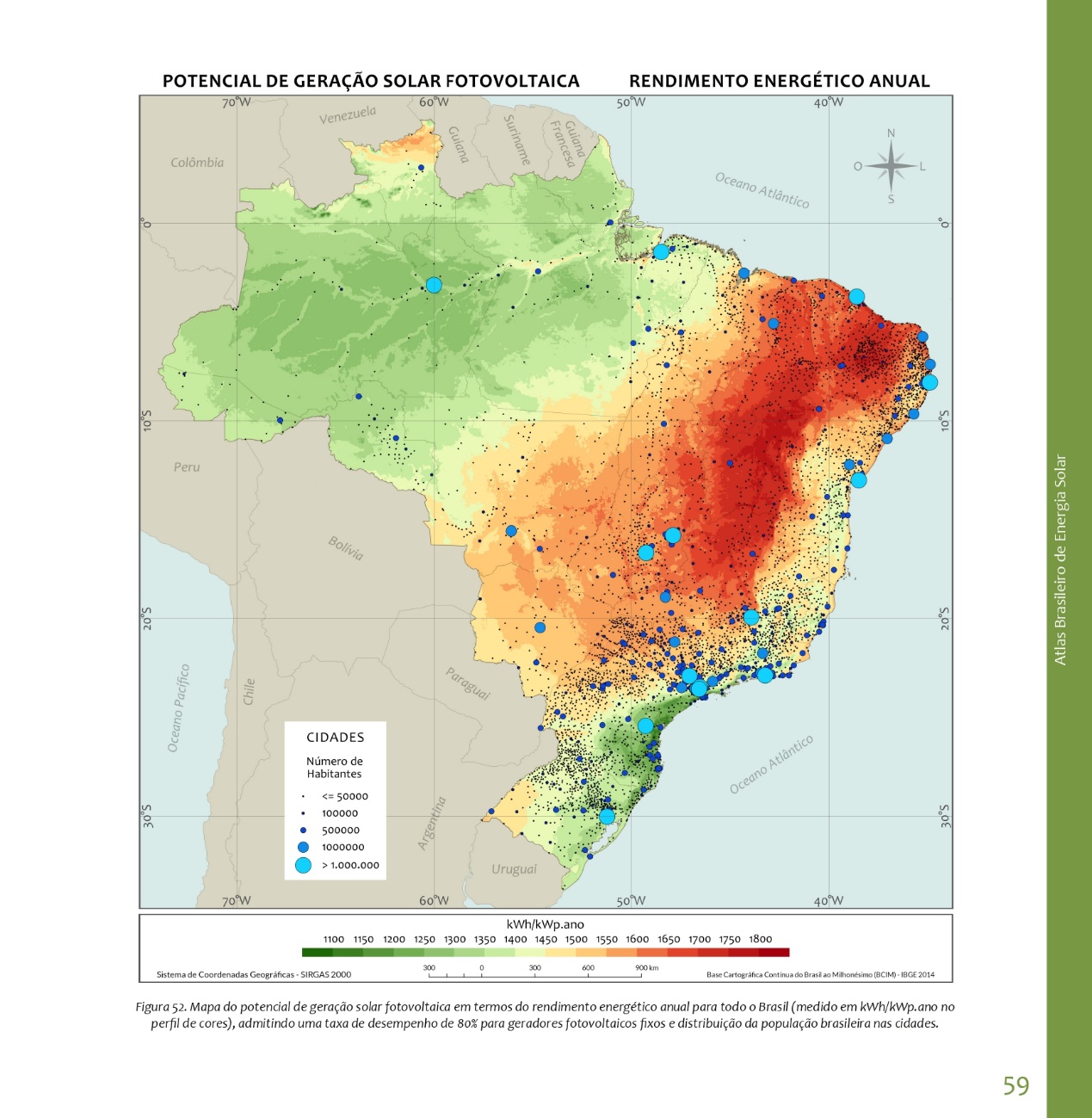
Sistemas supervisórios oferecem recursos avançados de detecção e diagnóstico de falhas. Ele pode monitorar constantemente o desempenho dos *Trackers*, detectando problemas, como mau funcionamento de sensores ou falhas mecânicas, e fornecendo alertas e notificações em tempo real e colocando a planta em estados de emergência caso preciso. Isso permite uma rápida intervenção e manutenção corretiva, minimizando o tempo de inatividade e garantindo a operação contínua e confiável do sistema. Com um sistema supervisório, as instalações solares podem alcançar todo o seu potencial, contribuindo para a transição para fontes de energia mais limpas e sustentáveis.

Com essa perspectiva de crescimento no uso de sistema de rastreamento e o consequente aumento da complexidade dos sistemas dos seus gerenciamentos, propõem-se o estudo e desenvolvimento de um sistema de controle SCADA (*Supervisory Control And Data Acquisition*) para sistemas de rastreamento solar *Tracker.* Um sistema SCADA deve possuir a capacidade de supervisionar, controlar e fazer a aquisição dos dados do sistema, que englobam valores de geração de energia, valores de sensoriamento e movimentação dos motores. Além disso, os sistemas de controle e supervisão devem contar com uma camada que garanta a segurança e integridade do sistema.

## Antecedente do problema

O Brasil é um país de grande área territorial, possuindo muitos climas e características próprias em cada região, possuindo predominantemente um clima tropical, semiárido e subtropical. Por possuir áreas que cortam desde a linha do equador até regiões subtropicais, a geração de energia solar torna-se um problema para as regiões mais afastadas da linha do equador, necessitando de sistemas mais inteligentes e capazes de gerar mais energia com menos irradiação solar. Na Figura 1 pode-se ver os níveis de irradiação anuais médios para irradiação solar normal no território brasileiro. Destacam-se duas regiões no mapa. A primeira região, a região de clima semiárido em vermelho, localizados aproximadamente em 10ºS 45ºW, que correspondem a zonas favoráveis à implementação de usinas de painéis solares fixos ou de um grau de liberdade como a de São Gonçalo. A segunda região localizada aproximadamente a 30ºS 54ºW, de clima subtropical correspondente ao estado do Rio Grande do Sul que será o ponto estudado.

Figura 1: Mapa do potencial de geração solar fotovoltaica em termos do rendimento energético anual para todo o Brasil (medido em kWh/kWp.ano )



Fonte: PEREIRA, E. B. (2017)

Devido a grandes extensões territoriais e estando situado em grande parte na região subtropical, o Brasil possui grande potencial de geração de energia solar fazendo uso de sistemas de rastreamento. Com essa perspectiva de crescimento, o uso de sistemas SCADA para monitoramento e supervisão desses sistemas, se torna um ponto importante que deve ser considerado no desenvolvimento dessas tecnologias.

## Descrição do problema

Devido ao complexo sistema de rastreamento, que permite o uso de motores de alta potência, inversor de frequência e uma gama de sensores para posicionamento, detecção de condições climáticas adversas, sistemas de controle em diferentes malhas e processamentos de dados, como valores de geração de energia dos painéis solares e consumo de energia por parte do sistema, o sistema *Tracker* se torna um sistema complexo para ser gerenciado. Dessa forma, a fim de viabilizar sistemas de rastreamento *Trackers* mais seguros e que permitam a fácil manutenção do sistema sem a necessidade de um serviço especializado, é necessário que se implemente um sistema SCADA que seja capaz de realizar esse gerenciamento e permita que um operador ou usuário final, esteja no controle do sistema sem que seja necessário um conhecimento técnico especializado.

## Objetivo

Tendo em vista o a necessidade de se implementar um sistema SCADA em sistemas de geração solar com rastreamento como forma de diminuir a complexidade do sistema, como descrito em tópicos anteriores e estimados os valores de produtividade de seguidores solares de dois graus de liberdade (2 eixos), propõem-se a criação de um sistema SCADA que seja capaz de supervisionar, realizar o controle e gerenciar o sistema como um todo.

O objetivo do presente trabalho não é o desenvolvimento do sistema de rastreamento solar, apenas o desenvolvimento de um *software* que seja capaz de supervisioná-lo. O sistema SCADA desenvolvido deverá ser capaz de se comunicar com o sistema de rastreamento através de um protocolo de comunicação a nível industrial, de forma a garantir a integridade e segurança na troca de informação por parte dos sistemas.

## Objetivos específicos

Dada a criação de um serviço SCADA personalizado, os seguintes objetivos específicos que o sistema SCADA terá, podem ser destacados:

1. Controle de acesso e níveis de permissão ao sistema (*login*);
2. Capacidade de ser integrado com sistemas *Tracker* que usem sensores e atuadores utilizando o protocolo industrial *Modbus RTU*;
3. Realizar a supervisão da geração e consumo do sistema;
4. Realizar o sensoriamento da posição dos motores.

Dessa forma, espera-se desenvolver um *software* SCADA utilizando o protocolo *Modbus* *RTU* utilizando a linguagem de programação *Python* com o *framework* de desenvolvimento de aplicações *Kivy* para personalização da aplicação. Esse sistema integrará com um sistema *Tracker* previamente desenvolvido ou então um sistema simulado, que possua os parâmetros de comunicação padronizados pelo protocolo *Modbus RTU*.

O aplicativo deverá contar com as funcionalidades de um sistema SCADA e possuir controles de acesso, níveis de controle e as demais funcionalidades de um sistema SCADA.

# revisão bibliográfica

Os sistemas SCADA (*Supervisory Control and Data Acquisition*) desempenham um papel essencial na automação e monitoramento de processos dentro da indústria. Esses sistemas desempenham o papel de coletar, controlar e analisar dados de uma planta em tempo real, permitindo a supervisão e o controle efetivo de sistemas complexos. Um sistema SCADA é uma solução tecnológica que integra *hardware* e *software* para fornecer controle e monitoramento centralizado de um ou mais processos, permitindo que dados sejam trocados entre os sistemas, além realizar as tarefas de coletar dados de dispositivos e sensores distribuídos em uma planta, instalação ou sistema, processar e apresentá-los de forma visual e compreensível para os operadores e tomadores de decisão, facilitando o trabalho de supervisão de sistemas complexos.

Nesse capitulo serão discutidos aspectos físicos de um sistema SCADA tais quais suas arquiteturas de *hardware*, comunicação entre sensores e atuadores, sistemas de proteção contra ataques, surtos dos sistemas, funcionalidades e o interfaceamento entre as operações e o operador do sistema, visando uma aplicação voltada um sistema de geração solar.

## O que é um sistema SCADA

De acordo com Junior (2019), um sistema de supervisão e controle consiste na atuação do homem nos processos de produção através e mecanismos confiáveis e que garantam um bom desempenho das ações, a segurança no controle de ambientes de difícil acesso humano e a minimização de falhas, que garante assim a otimização da produção.

Esses sistemas são usados em plantas e processos que possuam medições e controle de atuadores a longas distâncias, disponibilizando em uma única central, todas as informações desse processo, para que um ser humano, através desses dados, possa tomar ações de forma remota, reduzindo tempo nas operações.

São características importantes de um sistema SCADA as ações de: telecomando, telemedição e telesupervisão, que possibilitam que a planta seja monitorada de forma remota, a aquisição dos dados de sensores e a datação desses valores em bancos de dados para análises de processo.

Na Figura 2 é mostrado um esquema em formato de pirâmide apresentado por Santos, (sd), onde são relacionados a hierarquia entre os dispositivos de campo ou de instrumentação, os dispositivos de controle e dispositivos de supervisão remota.

Figura 2: Pirâmide de um sistema SCADA



Fonte: SANTOS, sd

Essa relação entre os dispositivos mostra a importância do emprego de um sistema SCADA, uma vez que ele consegue unificar uma grande variedade de informações de sensoriamento em uma central de operações.

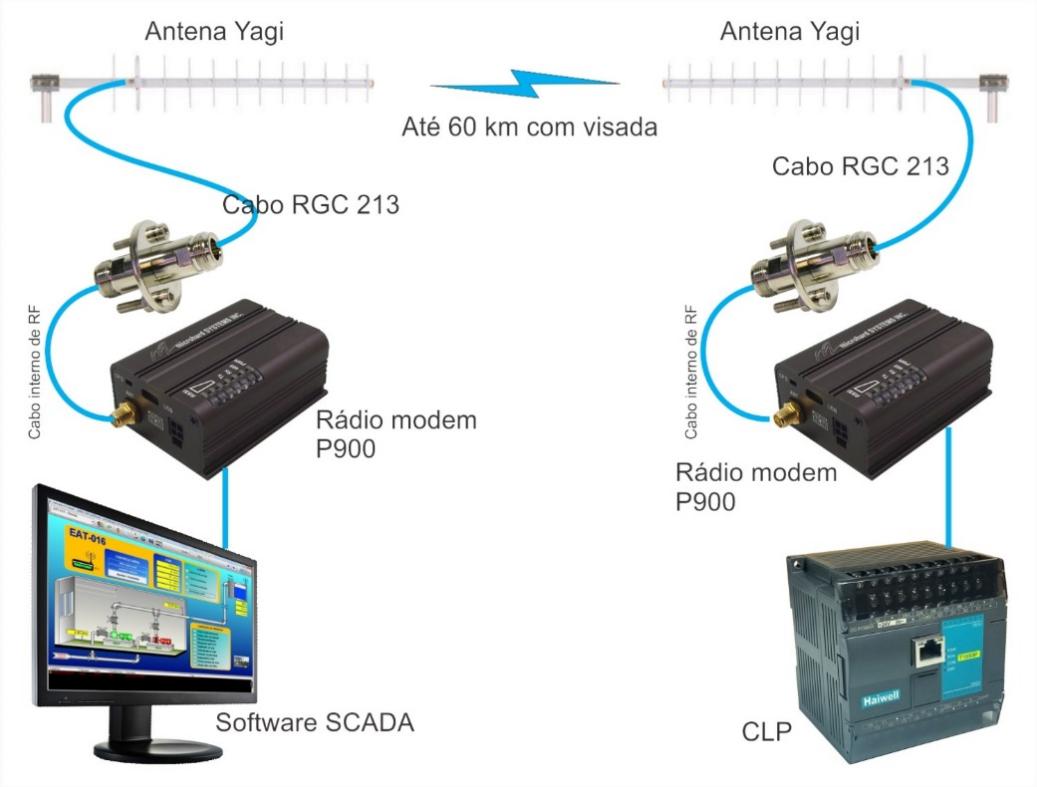
Uma das etapas de grande importância em um sistema SCADA está nos dispositivos que estabelecem a comunicação entre os dispositivos de campo e a central de operação. Os dispositivos que são responsáveis pela comunicação entre os sistemas são os Processadores de Fronteira (FEP do inglês *Front End Point*) e os MODEMs (Moduladores e Demoduladores).

Esses dispositivos são responsáveis pela transmissão dos dados trafegados pelas comunicações do sistema, que conseguem condicionar os sinais provenientes dos sensores em sinais elétricos tratados para envia-los à central de comando. No outro lado é recebido os sinais e os condicionam novamente para que os sistemas SCADA possa os utilizar.

Essas técnicas de conversão estão relacionadas com os MODEMs e estão fortemente relacionados ao meio físico que estão inseridos, podendo realizar as transmissões através do ar com o uso de rádio frequências por exemplo, ou em meio elétricos, utilizando protocolos específicos de comunicação, mas não se limitam apenas a esses dois métodos.

Na Figura 3 está representado um exemplo de aplicação dos MODEMs onde estabelecem a comunicação entre um CLP e um sistema SCADA. Nessa comunicação, os sinais são condicionados de um dispositivo para o outro através do uso dos MODEMs, que condicionam os sinais para serem transmitidos via Rádio.

Figura 3: Exemplo de MODEM de comunicação.



Fonte: Desconhecido.

## Protocolos de comunicação

Um dos elementos essenciais para garantir uma comunicação efetiva entre sistemas computacionais ou de automação é a utilização de protocolos de comunicação. De acordo com Junior (2019), um protocolo de comunicação consiste em um conjunto de procedimentos e regras que controlam e regulam a transferência de dados e a conexão entre os sistemas envolvidos.

Para estabelecer uma comunicação eficiente, é necessário contar com serviços de comunicação básicos que abrangem diferentes camadas do modelo de referência ISO/OSI. O modelo de referência de interconexões de sistemas abertos (OSI - *Open Systems Interconnection*) é normatizado pela organização internacional de padrões (ISO - *International Standards Organization*) e por este motivo é frequentemente chamado de modelo ISO/OSI. De acordo com Tanenbaum, (2011), o modelo ISO/OSI é composto por sete camadas, cada uma desempenhando funções específicas no processo de comunicação, sendo elas:

1. **Camada Física**: É a camada mais baixa do modelo e trata da transmissão física dos dados. Ela define as características elétricas, mecânicas e de transmissão dos sinais, incluindo a escolha do meio de transmissão, como cabos de cobre, fibras óticas ou ondas de rádio. Nessa camada, são especificados aspectos como a codificação dos dados e a topologia da rede.
2. **Camada de Enlace de Dados**: Responsável por garantir a transferência confiável dos dados entre dispositivos adjacentes na rede. Ela trata da detecção e correção de erros, controle de fluxo, controle de acesso ao meio físico e endereçamento físico. Essa camada divide os dados em quadros (*frames*) e os transmite de forma segura, estabelecendo uma conexão ponto a ponto.
3. **Camada de Rede**: Tem como objetivo realizar o roteamento dos pacotes de dados entre redes diferentes. Ela define os endereços lógicos dos dispositivos e implementa protocolos de roteamento, como o IP (*Internet Protocol*), para determinar a melhor rota para os pacotes. Nessa camada, ocorre o encapsulamento dos pacotes de dados em datagramas, que são enviados pela rede.
4. **Camada de Transporte**: Foca na transferência de dados fim a fim, estabelecendo uma conexão confiável e transparente entre os dispositivos de origem e destino. Ela oferece mecanismos para o controle de fluxo, controle de erros, segmentação e remontagem dos dados. Os principais protocolos utilizados nessa camada são o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*).
5. **Camada de Sessão**: Responsável pelo estabelecimento, gerenciamento e finalização das sessões de comunicação entre os aplicativos em execução nos dispositivos. Ela define os mecanismos para o controle de diálogo, sincronização e recuperação em caso de falhas.
6. **Camada de Apresentação**: Lida com a representação dos dados, garantindo que eles sejam compreendidos pelos dispositivos de destino. Ela trata da codificação, compressão e criptografia dos dados, além de definir os formatos de dados utilizados na comunicação.
7. **Camada de Aplicação**: É a camada mais alta do modelo e engloba os aplicativos e serviços que interagem diretamente com os usuários. Ela inclui protocolos específicos para serviços como e-mail (SMTP), transferência de arquivos (FTP), navegação na *web* (HTTP) por exemplo.

Na prática, seguindo esse modelo, a comunicação é facilitada devido a adoção de interfaces de comunicação padronizadas, que estabelecem desde a conexão física entre os dispositivos à aplicação. Algumas interfaces de comunicação físicas utilizadas para aplicações de controle supervisório comuns incluem o RS-232, RS-485, *Ethernet*, *Wi-Fi*, *LoRA*, entre outras. Cada uma dessas interfaces possui características e especificações próprias, adequadas a diferentes necessidades de comunicação e adotam pelo menos uma camada do modelo ISO/OSI.

Além das interfaces, os protocolos de aplicação desempenham um papel fundamental na organização e estruturação dos dados a serem comunicados. Esses protocolos são responsáveis por definir as regras de formatação, codificação, controle de erros, autenticação e outros aspectos relacionados à transferência de dados específicos de uma aplicação. Exemplos de protocolos de aplicação amplamente utilizados são o *Modbus* e o *Profibus*, que são frequentemente empregados em aplicações industriais para comunicação entre dispositivos.

## Aplicações

O protocolo *Modbus* é amplamente utilizado como uma aplicação de comunicação em sistemas de supervisão e controle industrial. Ele se baseia nas camadas mais baixas do modelo ISO/OSI, como a camada física e a camada de enlace de dados, e define um conjunto de regras para a troca de informações entre dispositivos mestre e escravo.

O *Modbus* permite a comunicação entre um dispositivo mestre (como um sistema de supervisão) e vários dispositivos escravos (como sensores, atuadores e controladores) conectados em uma rede. Ele utiliza mensagens simples para solicitar dados de leitura, escrita ou controle, utilizando endereços específicos para cada dispositivo escravo.

Uma das principais características do *Modbus* é a sua simplicidade e flexibilidade. Ele pode ser implementado em diferentes padrões de interfaceamento físico, como RS-232, RS-485 ou TCP/IP, e é compatível com uma ampla variedade de dispositivos e fabricantes. Além disso, ele suporta diferentes modos, como modo ASCII (utilizando caracteres ASCII) e modo RTU, adequando-se às necessidades específicas de cada aplicação.

Outro protocolo muito utilizado é o *Profibus* (*Process Field Bus*), utilizado em sistemas de automação industrial. O *Profibus* é um protocolo de comunicação em rede projetado para conectar dispositivos de campo, como sensores, atuadores e controladores, em um ambiente industrial.

Enquanto o *Modbus* é mais simples e adequado para aplicações menores e redes locais, o *Profibus* é mais robusto e é utilizado em sistemas de automação complexos, abrangendo diferentes níveis hierárquicos de controle. Ele suporta uma ampla variedade de dispositivos e oferece recursos avançados, como diagnóstico de falhas, controle de sincronização e comunicação em tempo real.

Em suma, os protocolos de comunicação *Modbus* e *Profibus* são exemplos de protocolos industriais que utilizam o modelo de referência ISO/OSI como base para estabelecer a comunicação em um sistema de supervisão e controle. Ambos os protocolos se encaixam nas camadas de aplicação do modelo ISO/OSI, permitindo a troca de dados entre dispositivos mestre e escravo em um ambiente industrial.

O *Modbus* é conhecido por sua simplicidade e flexibilidade, sendo amplamente adotado em aplicações de menor escala. Ele oferece recursos básicos de leitura, escrita e controle de dispositivos, sendo adequado para comunicação local.

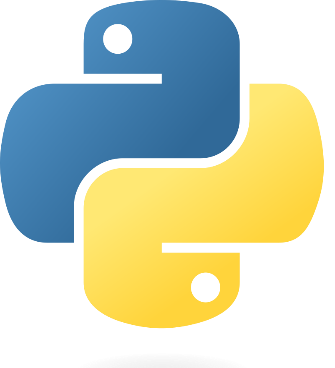
# MATERIAIS E MÉTODOS

Neste tópico serão abordados os materiais e métodos utilizados para o desenvolvimento do sistema SCADA, tais linguagens de programação foram utilizadas. *Frameworks* para trabalho e desenvolvimento dos *softwares* da aplicação e infraestrutura de processamento dos dados com detalhamento da unidade de processamento da aplicação e interface homem-máquina previsto.

## Especificações de *Software* - Python 3.10

O projeto da interface fora desenvolvido inteiramente utilizando a linguagem de programação *Python* na versão 3.10.0 disponível gratuitamente para *download* em <https://www.python.org/downloads/> sobre a identificação de *download* “tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18” (Figura 4).

Figura 4: Logo da linguagem de programação Python.



Fonte: Disponível em: www.python.org/

A linguagem de programação *Python* é uma linguagem de alto nível que se destaca por sua simplicidade e legibilidade. Ela foi criada por *Guido van Rossum* e lançada pela primeira vez em 1991, com o objetivo de ser uma linguagem fácil de aprender e usar. Desde então, a linguagem se tornou uma das mais populares e amplamente adotadas em diversos campos, como desenvolvimento *web*, ciência de dados, automação de tarefas, inteligência artificial e recentemente, no desenvolvimento de aplicativos e aplicações *mobile*.

Uma das características distintivas do *Python* é seu modelo de programação que enfatiza a legibilidade do código. A sintaxe limpa e clara da linguagem, juntamente com sua abordagem de indentação significativa, facilita a compreensão do código.

Além disso, o *Python* é uma linguagem de programação interpretada, ao contrário de outras linguagens populares como C, o que significa que não requer um processo de compilação antes da execução. Isso permite uma prototipação rápida e uma abordagem interativa no desenvolvimento de *software*. Além disso, a linguagem suporta múltiplos paradigmas de programação, incluindo programação procedural, orientada a objetos e funcional, fornecendo flexibilidade aos desenvolvedores para escolher a abordagem mais adequada para as suas necessidades.

A popularidade do *Python* tem crescido significativamente nos últimos anos. Apesar de ter sido criada em 1991, somente nos últimos anos ela começou a tomar espaço dentre as linguagens mais famosas, isso se deve em parte à sua comunidade ativa e engajada, que contribui com bibliotecas e *frameworks* de código aberto para desenvolvimento *Python*, suportando a comunicação com protocolos de rede e desenvolvimento de interfaces gráficas.

Por esses motivos, a linguagem de programação foi utilizada para o desenvolvimento da aplicação SCADA, sendo utilizada em todas as etapas do projeto, desde etapas de aquisição de dados do sistema, integração de *drivers* de rede, até o desenvolvimento gráfico das aplicações através de *frameworks* específicos para criação de aplicações gráficas.

Pela sua versatilidade, a linguagem permite que *drivers* de comunicação como os do protocolo *Modbus* sejam executados em tempo de execução, gerenciando o tráfego de dados via serial. Além disso, aplicações que usam armazenamento de informações em bancos de dados podem ser desenvolvidas diretamente na linguagem.

Devido às características dessas funcionalidades, serão discutidos individualmente cada *framework* utilizado no desenvolvimento da aplicação.

## Comunicação de dados – *Modbus*

O protocolo de comunicação *Modbus* é um protocolo amplamente utilizado na indústria para trocar informações entre dispositivos eletrônicos. Desenvolvido na década de 70, ele se tornou um padrão de fato para a comunicação entre dispositivos de controle e monitoramento em sistemas de automação industrial, estando fortemente presente em indústrias atualmente.

O protocolo de comunicações Modbus foi criado inicialmente para ser usado nos CLPs (Controladores Lógicos Programáveis) da *Modicon* (vide Figura 5).

Figura 5: Primeiro CLP da Modcon - MODICON 084



Fonte: Disponível em: <https://www.linkedin.com/pulse/voc%C3%AA-j%C3%A1-ouviu-falar-na-rede-modbus-rodrigo-moreira-borges/?originalSubdomain=pt>

A empresa *Modicon* foi adquirida pela *Schneider Electric*, resultando em uma mudança significativa no cenário da indústria de automação e controle. Com essa aquisição, o protocolo *Modbus*, utilizado para a comunicação entre dispositivos de controle e automação industrial, alcançou um *status* de ampla disseminação.

Desde então, a manutenção e desenvolvimento contínuo do protocolo têm sido responsabilidade da entidade conhecida como *Modbus Organization* (vide Figura 6).

Figura 6: Logo da ModBus Organization.



**Fonte: Disponível em: https://www.linkedin.com/pulse/voc%C3%AA-j%C3%A1-ouviu-falar-na-rede-modbus-rodrigo-moreira-borges/?originalSubdomain=pt**

Em abril de 2004, a *Schneider Electric* transferiu os direitos do protocolo para essa organização, que se tornou uma associação de usuários e fabricantes comprometidos em promover e preservar a utilização do protocolo *Modbus*. A *Modbus Organization* desempenha um papel vital na defesa da continuidade do uso do protocolo, garantindo sua atualização e interoperabilidade entre os dispositivos compatíveis. Com seu compromisso em manter o protocolo atualizado e relevante para as necessidades da indústria, a *Modbus Organization* desempenha um papel crucial na promoção da comunicação eficiente e confiável entre dispositivos industriais.

Graças a esses esforços, o protocolo de comunicação *Modbus* se mantém até os dias de hoje, adaptando-se aos propósitos da indústria e conseguindo entregar um sistema robusto e confiável. O padrão passou por diversas modificações e adaptações para usos específicos, existindo 3 variações do protocolo gratuitos e disponíveis atualmente, sendo eles:

1. Modbus RTU
2. Modbus ASCII
3. Modbus TCP/IP

### Estrutura dos pacotes *Modbus*

Todos modelos de comunicação seguem o mesmo propósito e possuem muitas similaridades entre eles. A fim de manter essa compatibilidade, o modelo *Modbus* divide seus pacotes em ADU (*Application Data Unit*), que diferencia os pacotes pelo tipo dentre os 3 modelos mostrados acima e engloba o PDU (*Protocol Data Unit*). Essa definição fica mais clara na Figura 7.

Figura 7: Modelo de separação dos campos de dados no protocolo Modbus.



Fonte: Disponível em <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>

Essa divisão é feita para permitir a flexibilidade e a compatibilidade do protocolo Modbus em diferentes camadas de comunicação.

O PDU é a parte central da mensagem *Modbus* e contém as informações específicas da função a ser executada. Ele inclui o endereço do dispositivo, a função a ser executada e os dados associados à função. O PDU é independente da camada de transporte ou meio físico de comunicação. Já o ADU é a estrutura completa da mensagem *Modbus*, incluindo o PDU e informações adicionais necessárias para a transmissão e recepção dos dados. Ele encapsula o PDU em uma estrutura de quadro (*frame*) adequada para a camada de transporte.

Na

Figura 8 pode-se comparar dois pacotes de dados de dois modelos *Modbus*, o RTU e TCP/IP:

Figura 8: Pacotes *Modbus* RTU e TCP/IP.



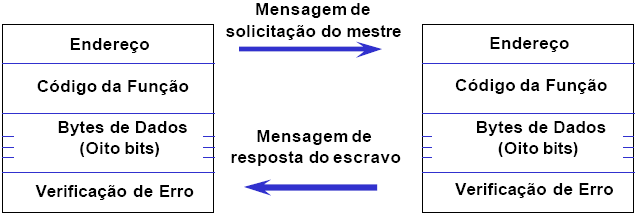
Fonte: Disponível em <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>

Percebe-se na Figura 8 em verde, os campos de dados destinados ao PDU, sendo esse, igual nos dois protocolos RTU/ASCII e TCP/IP. Já em roxo se tem o ADU que diferencia os protocolos. No *Modbus* RTU, o PDU é encapsulado em um quadro que contém informações de início e parada de bits, bem como informações de controle de fluxo. O quadro é transmitido por meio de uma conexão serial síncrona (como RS-485 ou RS-232). No *Modbus* ASCII, o PDU é encapsulado em uma estrutura de quadro que usa caracteres ASCII para representar os dados. Ele também contém informações de início e fim de quadro. O quadro é transmitido por meio de uma conexão serial assíncrona. No Modbus TCP/IP, o PDU é encapsulado em pacotes TCP/IP, onde o ADU inclui informações de endereço IP e número de porta para direcionar a mensagem ao dispositivo escravo correto.

### Modelo de comunicação mestre-escravo

Cada pacote *Modbus* possui claramente as informações de destino e o tipo de solicitação. Essa estrutura permite que o *Modbus* seja utilizado em um modelo de Mestre-Escravo, onde os dispositivos de controle solicitam mensagens aos dispositivos de campo e esses respondem por sua vez de acordo com o tipo de mensagem do mestre (vide Figura 9).

Figura 9: Modelo mestre-escravo *modbus*.



Fonte: Disponível em <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>

### Código de função

O FC (Código de Função do inglês *Funtion Code*) auxilia o escravo a executar a ação requerida pelo mestre. Algumas funções podem ser:

* Endereços dos registradores (registro inicial);
* Quantidade de registros a serem lidos;
* Contador da quantidade de *bytes* no campo de dados;
* O campo de dados pode não existir. Neste caso o próprio código da função sozinho especifica a ação requerida;
* Se não ocorrer nenhum erro na função especificada na requisição, a resposta do escravo conterá o dado requisitado, caso contrário o campo dado conterá um código de exceção;

Quando respeitada essa requisição, o escravo retorna uma mensagem de resposta também formata de um modo que o mestre entenda. Nesse modelo, o mestre pode ser entendido com um cliente, enquanto o dispositivo de campo é o servidor do sistema (vide Figura 10).

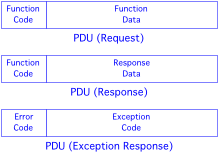
Figura 10: Modelo de requisição e resposta Modbus.



Fonte:Disponível em <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>

Dentro dos FC existem os códigos de função e os códigos de exceção. Os códigos de função variam do código 1 ao 127, enquanto que exceções variam de 128 a 255. As exceções são tipicamente codificadas em códigos de função FC + 128. Para cada função, ou seja, requisição de serviços do mestre, existe um código de função no *Modbus*. Assim há 3 tipos de PDUs como ilustra a Figura 11.

Figura 11: Modelos de PDUs do modelo *Modbus*.

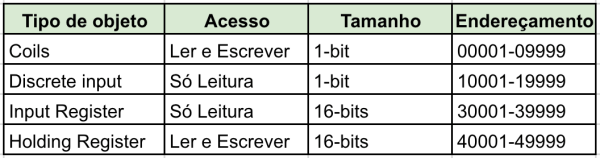


Fonte: Disponível em <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>

### Tipos de registradores

Os registradores *Modbus* são elementos fundamentais do protocolo Modbus e são usados para armazenar e acessar dados em dispositivos escravos (servidores) em uma rede *Modbus*. Existem 4 tipos principais de registradores como mostrados na Figura 12.

Figura 12: Registradores Modbus.



Fonte: Disponível em <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>

1. *Input Registers* (IR): Os *Input* *Registers* são usados para armazenar dados somente leitura. Eles contêm informações que podem ser lidas pelo dispositivo mestre (cliente), como valores de sensores, estados de dispositivos ou qualquer outro dado que não possa ser alterado pelo mestre.
2. *Holding* *Registers* (HR): Os *Holding* *Registers* são usados para armazenar dados que podem ser lidos e escritos pelo dispositivo mestre. Eles podem conter informações como configurações de dispositivos, valores de controle ou qualquer dado que possa ser modificado pelo mestre.
3. *Input* *Discrete* (ID): Os *Input* *Discrete* são usados para representar entradas discretas no dispositivo escravo. Eles representam estados lógicos, como sensores binários ou interruptores, e são apenas para leitura.
4. *Coils*: Os *Coils* são semelhantes aos *Input* *Discrete*, mas podem ser lidos e escritos pelo dispositivo mestre. Eles são frequentemente usados para controlar dispositivos binários, como relés ou válvulas.

Cada registrador é identificado por um endereço único dentro do dispositivo escravo, permitindo que o mestre especifique qual registrador deseja ler ou escrever. Os registradores *Modbus* são organizados em tabelas, com cada tabela correspondendo a um tipo de registrador específico. A quantidade de registradores disponíveis em um dispositivo *Modbus* depende de sua capacidade e configuração específicas. Geralmente, os dispositivos *Modbus* têm uma quantidade limitada de registradores, com tamanhos e endereços pré-definidos.

Devido o fácil acesso desses registradores e organização, se torna fácil administrar um processo, pois os dados estarão armazenados de uma maneira padronizada, promovendo uma interoperabilidade entre dispositivos.

Para a aplicação SCADA em particular, as propriedades da comunicação *Modbus* *RTU* satisfazem muito bem os requisitos, uma vez que o sistema estará conectado próximo ao sistema e pode ser feito utilizando uma comunicação serial. Além disso, a linguagem de programação *Python* possui bibliotecas criadas especificamente para esse tipo de comunicação como o caso das bibliotecas:

1. ***pymodbus***: É uma biblioteca que implementa os protocolos Modbus RTU, TCP e UDP. Ela fornece uma API simples e abstrata para interagir com dispositivos *Modbus* e suporta tanto a escrita quanto a leitura de registradores *Modbus*.
2. **minimalmodbus**: É uma biblioteca que oferece suporte a comunicação *Modbus* *RTU* exclusivamente por meio de uma porta serial. Ela é fácil de usar e fornece métodos para ler e escrever registradores *Modbus*, bem como configurações adicionais, como endereço do dispositivo, paridade, entre outros.
3. ***pyModbusTCP***: É uma biblioteca que implementa o protocolo *Modbus* *TCP* exclusivamente. Ela permite a comunicação com dispositivos *Modbus* por meio de uma rede TCP/IP, oferecendo métodos para ler e escrever registradores *Modbus*.

## Interface gráfica - *Kivy*

O *framework* escolhido para o desenvolvimento da interface homem máquina também chamados de GUI (Interface Gráfica de Usuário do inglês *Graphical User Interface*) foi o *kivy*. O *Kivy* é um *framework* de código aberto escrito em *Python* que permite o desenvolvimento de aplicativos multiplataforma com interfaces gráficas voltadas para o usuário. Ele foi projetado para criar aplicativos que podem ser executados em uma variedade de plataformas, incluindo computadores pessoais, dispositivos móveis como *smartphones* e até mesmo plataformas independentes como o caso da *Raspberry Pi,* plataforma usada para o processamento da aplicação que será descrita em tópicos futuros.

O *Kivy* possui uma arquitetura flexível que segue padrões de desenvolvimento de código limpo como o padrão de desenvolvimento MVC (*Model-View-Controller*), permitindo a separação clara entre a lógica de negócios e a apresentação visual da aplicação. Isso facilita a manutenção do código, o reuso de componentes e a escalabilidade do aplicativo, sendo ideal para o desenvolvimento da aplicação SCADA, uma vez que permite separar com clareza quais as regras de negócio entre a aplicação e o sistema supervisionado, permitindo que o programa rode de forma independente, blocos de código relacionados a comunicação com o sistema e blocos de código relacionados a renderização da interface gráfica, tornando o aplicativo mais robusto contra falhas.

Além disso, o *framework* *Kivy* conta com uma série de expansões de código aberto como o *KivyMD*, uma extensão que adiciona componentes gráficos relacionados ao termo “*Material Design”*, que é um conjunto de diretrizes padronizados pela empresa *Google* para a criação das suas interfaces gráficas. Dessa forma o *KivyMD* permite que as interfaces criadas possuam um estilo agradável, uma vez que padroniza as formas do aplicativo, cores, tipografia e animações em tela.

O modelo de programação em *kivy* permite que sejam criadas interfaces utilizando preceitos muito bem difundidos em outras linguagens de programação utilizadas para a criação de interfaces gráficas, como o *Java* por exemplo. O *Java* é uma linguagem de programação orientada a objetos muito popular na criação de interfaces, isso esta ligado ao fato de que dentro de uma interface, existe uma hierarquia dos componentes mostrados em tela e muitos deles são objetos similares, que podem ser condensados em um objeto único, chamado de objeto de interface.

Dessa forma, um objeto de interface pode ser utilizado em diversos lugares com diversas funções, sem perder suas características fundamentais. Além disso, ele pode herdar informações de outros objetos de interface que possuem uma hierarquia maior que a sua, sendo essa uma técnica chamada de herança. Na Figura 13 é possível ver esse modelo de heranças acontecendo.

Figura 13: Modelo de hierarquia de objetos em interfaces orientadas a objetos.

Fonte: (ASSIS, 2012)

## Banco de dados - *SQLite3*

Outro ponto fundamental para a aplicação está no desenvolvimento de um banco de dados que irá armazenar as informações do sistema. A necessidade de um banco de dados robusto é fundamental para garantir a integridade dos registros feitos pelos sensores do sistema. Dentro do escopo de programação em *Pyhton*, uma boa solução para implementar um banco de dados é o *SQLite3*, uma biblioteca padrão da linguagem que possui um sistema de gerenciamento de dados leve e autônomo, utilizando de arquivos locais, sem a necessidade de um servidor dedicado para banco de dados.

Dentre as vantagens do *sqlite3* estão:

1. O *SQLite3* já está incluído na instalação padrão do *Python*, portanto, não é necessário instalar bibliotecas adicionais, basta importá-lo para usar.
2. Permite a criação de bancos de dados diretamente em arquivos locais, sem a necessidade de um servidor de banco de dados separado.
3. Suporta transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), garantindo a integridade dos dados mesmo em casos de falhas ou interrupções do sistema.
4. Foi projetado para ser rápido e eficiente, com boa performance mesmo em grandes conjuntos de dados.
5. É compatível com diferentes sistemas operacionais, incluindo *Windows*, *macOS* e *Linux*. Isso torna os aplicativos desenvolvidos com o *SQLite3* portáteis e independentes da plataforma.

Dessa forma é possível se criar tabelas de dados diretamente em código, permitindo que informações do sistema sejam armazenados de forma integra e acessíveis para o gerenciamento do sistema e criação de relatórios para avaliações de desempenho e geração.

## Decisão

A integração dos módulos *Modbus*, *Kivy* e *SQLite3* para o desenvolvimento do projeto SCADA em *Python* se mostrou uma boa opção devido a facilidade do uso das ferramentas e a forma com que os módulos tem de conversarem entre si, permitindo que haja um fluxo de informações e modularização do sistema, onde cada módulo é responsável por uma tarefa dentro da aplicação.

O módulo *Modbus* permite a comunicação com dispositivos de automação industrial, permitindo a leitura e escrita de dados nos registradores *Modbus*. Com o *Modbus*, é possível estabelecer a comunicação com controladores programáveis, sensores e atuadores, obtendo informações em tempo real sobre o estado dos equipamentos e realizando ações de controle quando necessário. Além disso, ele possui uma estrutura simples para leitura e escrita dos seus registradores, permitindo uma fácil adaptação do protocolo para o uso na aplicação especifica.

O *Kivy*, por sua vez, é um *framework* de interface gráfica de usuário multi-plataforma que permite o desenvolvimento de interfaces interativas e intuitivas para aplicativos. Com o *Kivy* e sua extensão *KivyMD*, é possível criar telas, gráficos, botões e outros elementos visuais de forma bonita e agradável ao uso, para visualizar e interagir com os dados do sistema SCADA de forma personalizável.

Por fim, o *SQLite3* é um sistema de gerenciamento de banco de dados embutido que permite armazenar e consultar dados coletados pelo sistema SCADA. Ele oferece recursos para criar tabelas, inserir e atualizar registros, além de executar consultas *SQL* para análise e relatórios. O *SQLite3* é adequado para aplicações SCADA menores e de médio porte, permitindo armazenar dados históricos, configurações e outros dados relevantes para o sistema de supervisão, sendo perfeito para aplicação em especifico, uma vez que é extremamente leve e otimizado quando comparado a outros bancos de dados, permitindo que ele seja utilizado em plataformas como *Raspberry Pi*.

Ao integrar esses três módulos em um desenvolvimento SCADA em *Python*, é possível criar um sistema completo para monitorar, controlar e armazenar dados do *Tracker*.

## Especificações de Hardware

### Raspberry Pi 3

A *Raspberry Pi* é uma série de computadores de placa única (do inglês *single-board computer*) desenvolvida pela *Raspberry Pi Foundation*. A *Raspberry Pi Foundation* é uma organização sem fins lucrativos com sede no Reino Unido, cujo objetivo é promover o ensino da ciência da computação e a capacitação tecnológica para pessoas de todas as idades ao redor do mundo.

O *Raspberry Pi B3* por sua vez, é um modelo de computador de placa única que oferece um ambiente de baixo custo, baixo consumo de energia e tamanho compacto. Ele é amplamente utilizado para projetos de automação, incluindo aplicações SCADA devido as suas especificações.

Em termos de *hardware* o *Raspberry Pi 3 model B* tem muito poder de processamento graças ao processador *Broadcom* *BCM2837* de 64*bits* e *clock* de 1,2GHz 64-*bit* *quad-core ARMv8* CPU, 1 GB de RAM, *Bluetooth* 4.1 e possui um controlador *Ethernet* dedicado. Possui quatro portas USB. Além disso, ele permite que seja conectado um HD externo nas suas portas USB, ou então possui uma entrada de cartão SD para armazenamento de dados e armazenamento do sistema operacional.

Além disso, uma das maiores vantagens do *Raspberry Pi* é que ele possui saída para conectar em monitores externos através de um cabo com interface HDMI (*High-Definition Multimedia Interface*). O Cabo HDMI faz a transmissão de dados em alta resolução, como por exemplo, computadores, televisores e videogames. A transmissão ocorre de forma simultânea para vídeo e áudio em altíssima qualidade.

### IHM

Para a visualização do processos e interatividade com a interface gráfica, fora usado um *display* de 11,6 polegadas de 1920x1080 *pixels*, compatível com HDMI e tela LEDIPS de forma compatível com as especificações de uso com o *Raspberry Pi* *3B* (vide Figura 14).

Figura 14: *Display* utilizado como IHM.



Fonte: Disponível em: <https://ae01.alicdn.com/kf/H644aca9d52b5427d89a8513404941302k/11-6-Polegada-1920x1080-Compat-vel-com-HDMI-1080P-Tela-LED-IPS-Monitor-HD-Port-til.jpg>

# DESENVOLVIMENTO

Na seguinte seção, será apresentado as técnicas utilizadas durante e o desenvolvimento do projeto de *software* da interface de controle. Esse tópico estará dividido nas seguintes áreas de desenvolvimento: s*oftwares* de controle, Interfaceamento com protocolo *Modbus* e sistema físico.

Na área do *software* de controle, serão apresentado técnicas utilizadas para se criar o modelo de supervisão utilizando os *frameworks* apresentados na seção 3. Na área de interfaceamento com o protocolo de comunicação *Modbus*, será apresentado como o *software* consegue comunicar com os dispositivos *Modbus* conectados no sistema. Essa comunicação se dá via *Drivers* de comunicação. Na ultima área será apresentado brevemente o sistema físico utilizado para o projeto e apresentado aspectos construtivos.

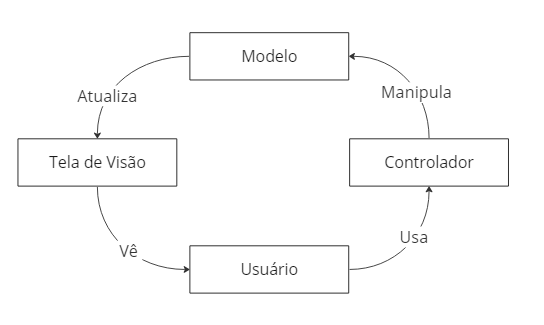
## Software de controle

Como apresentado nas subseções 3.1 a 3.3, a interface de controle fora construída inteiramente utilizando a linguagem de programação *Python* e *frameworks* de desenvolvimento tais quais *Kivy* e *KivyMD*. A escolha dessas ferramentas se dá devido a facilidade de se criar sistemas e objetos visuais rodando em uma IHM.

### Organização do Software de interface

O primeiro ponto levado em consideração na hora de se desenvolver a interface, esteve no modelo que os arquivos iriam estar arquitetados dentro do repositório do projeto. No *KivyMD* existe uma função denominada criação de projeto (*Create\_project* em inglês) que serve para iniciar um repositório organizado dentro de um modelo de escolha. O modelo de arquitetura escolhido foi o MVC (*Model-View-Controller*) (vide Figura 15) que significa a hierarquia de Modelo, Visão e Controlador, onde:

Figura 15: Hierarquia do modelo de organização MVC



Fonte: Próprio autor.

**Modelo**: Responsável pela manipulação de dados sensíveis e informações armazenadas em bancos de dados. Em geral, essa parte da aplicação não deve ser acessível diretamente pelo usuário e apenas o controlador deve possuir acesso a ela. É nesse espaço do projeto que todas as manipulações de *login*, cadastro de usuários e manipulações de níveis de acesso são realizadas. O controlador tem o papel de atualizar as informações contidas no Modelo e ele por sua vez tem o papel de atualizar a interface gráfica das mudanças ocorridas por ele.

**Controlador**: É o responsável pelo gerenciamento do projeto como um todo. Todas as manipulações de dados da interface, deverão passar pelo controlador. Ele é a interface entre o modelo e a tela de visão onde o usuário irá ter acesso. Toda interação do usuário com a interface de controle irá passar pelo controlador.

**Tela de visão:** É a interface entre o usuário e o sistema físico. Nessa área todas as definições gráficas são criadas e pensadas para que o usuário do sistema (os operadores) tenha a melhor experiência possível de uso, garantindo que ela seja fácil de entender e usar.

Utilizando dessa arquitetura, fica fácil definir as regras de negócio que devem ser implementadas no sistema, definido as principais funções do sistema e conseguindo realiza-las de forma linear.

### Processos internos

Um ponto importante durante a implementação esteve na criação dos processos internos da interface de controle, o chamado *backend* da aplicação. Esses processos estão intimamente relacionados à área do Controlador dentro do modelo MVC. Como toda manipulação da interface corre nos domínios do controlador, esse é responsável pelo gerenciamento da aplicação e possui um papel fundamental dentro da aplicação.

Dentre as responsabilidades do *backend* estão:

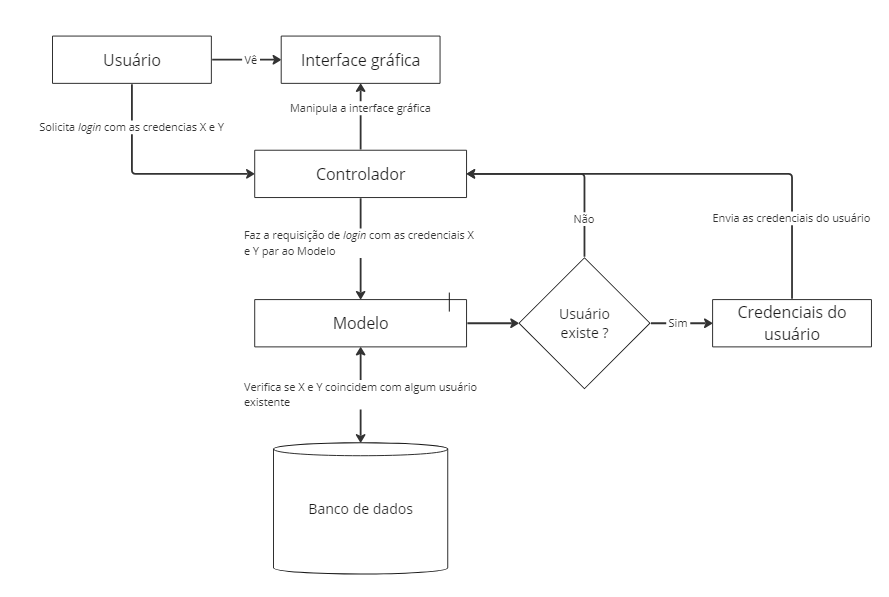
1. Gerenciar *logins* criação de novos usuários;
2. Garantir que usuário sem nível de acesso correto manipulem áreas sensíveis.

Nesse ponto é importante diferenciar as tarefas do Modelo às do Controlador, uma vez que o Controlador manda solicitações de login para o Modelo, mas não manipula os dados sensíveis diretamente, sendo esse um processo interno do Modelo.

#### Gerenciamento de logins e criação de usuários

Para o processo de gerenciamento de *logins*, na Figura 16 é possível ver a ordem de requisição realizada pelo Controlador.

Figura 16: Modelo de requisição de *login*.



Fonte: Próprio autor.

Dessa forma, percebe-se que o Controlador apenas intermedia a tela de interação com o usuário e o banco de dados usado pelo sistema, garantindo que o usuário não manipule dados sensíveis.

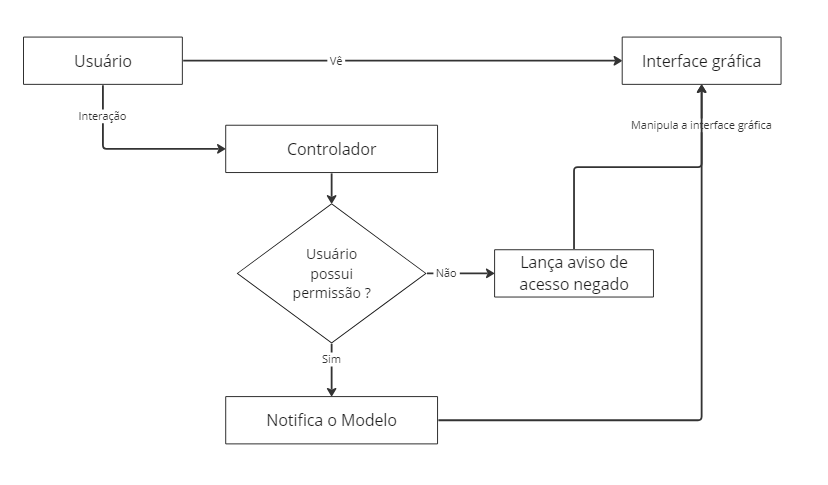
Para o gerenciamento de criação de novos usuários, o fluxograma do processo é semelhante ao processo de *login*, no entanto a solicitação do Controlador ao Modelo é a solicitação de registro.

#### Controle do nível de acesso a usuários

Sem dúvidas, uma das tarefas mais importantes dentro do sistema, é o controle dos níveis de acesso que cada usuário possui, a fim de garantir que somente usuários confiáveis possam manipular as partes sensíveis do sistema, tais quais: manipulação das rotinas de trabalho do sistema, credenciamento de novos usuários ou manipulação indevida das variáveis de funcionamento do sistema físico.

Dessa forma, toda vez que o usuário da interface realizar uma interação em zonas com proteção de nível de acesso, é papel do controlador verificar se o usuário possui tais permissões ou não para realizar essas ações. A Figura 17 mostra o fluxograma do processo de acesso a regiões do supervisório que possuem proteção de nível de acesso.

Figura 17: Fluxograma do processo de controle do nível de acesso



Fonte: Próprio autor.

Pode-se perceber que o controlador é o responsável pela verificação do nível de acesso do usuário que executou o *login*, uma vez que possuir as credenciais de *login*. No entanto, ele ainda possui a obrigação de notificar o Modelo para toda requisição ocorrida.

### Modelo

O Modelo representa a lógica de negócios e os dados subjacentes ao aplicativo. Ele contém a lógica para recuperar, armazenar e manipular os dados, bem como implementar as regras de negócios. O Modelo é independente da interface do usuário e da interação do usuário. Ele notifica as outras partes do padrão sobre mudanças em seus dados, geralmente usando o padrão *Observer* (Observador).

No contexto do sistema *Tracker*, o Modelo desempenha um papel crucial ao lidar com as comunicações com o sistema utilizando o protocolo *Modbus* e estabelecendo conexões de *login* através de *sockets* TCP/IP com um servidor SQL disponível para a aplicação. Além disso, o Modelo é responsável por gerenciar os níveis de acesso internos do sistema, definido os limites dos usuários.

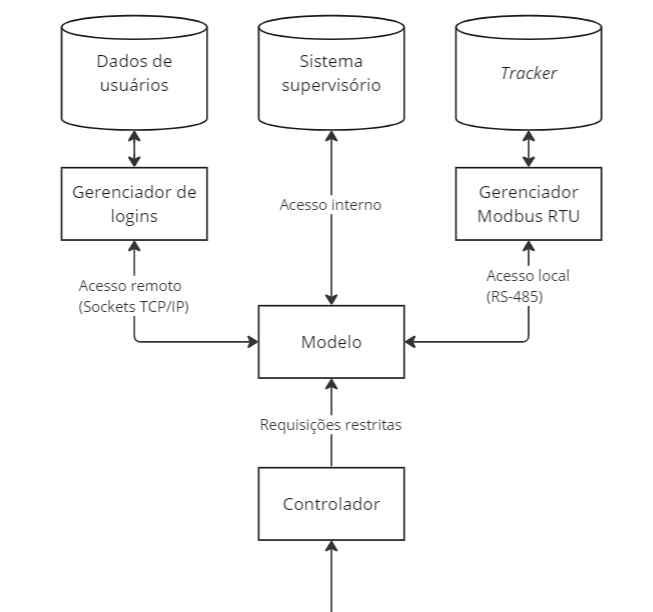
No que diz respeito às comunicações *Modbus*, o Modelo é encarregado de estabelecer a comunicação com os dispositivos do sistema *Tracker*, enviando e recebendo dados por meio desse protocolo e fazendo o interfaceamento ao meio físico RS-485 que é estabelecido pelo *Modbus RTU*. Ele implementa as funcionalidades necessárias para enviar comandos e receber respostas dos dispositivos, garantindo a integridade e a confiabilidade das comunicações *Modbus,* de forma que o sistema somente seja acessível através dele.

Quanto às conexões de *login* usando *sockets* TCP/IP e o servidor SQL, o Modelo irá gerenciar a autenticação e a autorização dos usuários. Ele verifica as credenciais fornecidas pelos usuários durante o processo de *login*, interage com o servidor SQL para validar as informações e determinar os níveis de acesso concedidos. Além disso, o Modelo mantém um controle interno dos níveis de acesso dos usuários, permitindo ou negando o acesso a recursos e funcionalidades específicas do sistema.

Dessa forma, o Modelo implementa as funcionalidades necessárias para estabelecer as conexões com dispositivos *Modbus*, realizar a autenticação dos usuários e gerenciar os níveis de acesso, garantindo um funcionamento adequado e seguro do sistema como um todo, garantindo que as regiões de acesso a dados sensíveis, sejam protegidas do usuário do sistema supervisório.

Na Figura 18 é possível se verificar o fluxograma dos processos que o Modelo executa e as relações que possui com cada etapa.

Figura 18: Fluxograma das responsabilidades do Modelo.



Fonte: Próprio autor.

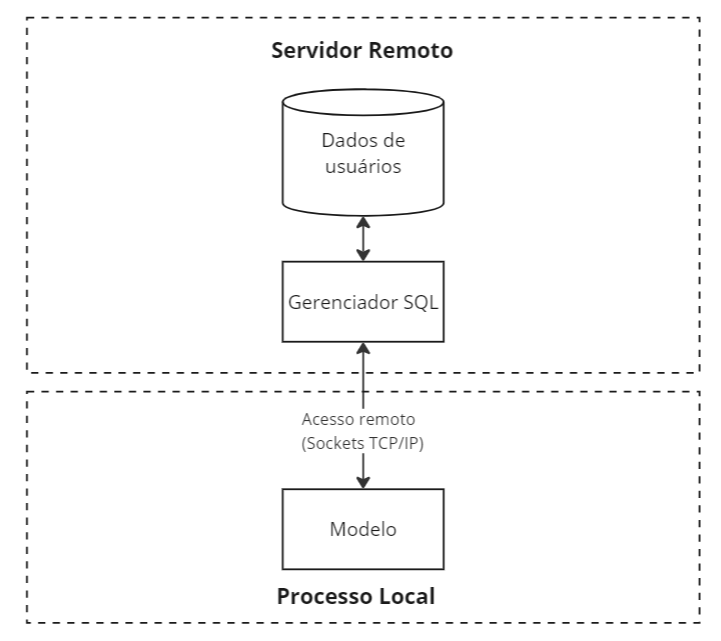
#### Autenticações de login

Para realizar o processo de autenticações de *logins*, foi utilizado um serviço de autenticação a qual um servidor remoto é acessado via pacotes *sockets* comcomunicação *TCP/IP,* onde pode-se garantir a entrega dos pacotes e estabelecer uma segurança na comunicação, através de métodos de criptografia. Na Figura 19 pode-se ver o fluxo de informações entre o Modelo e o servidor remoto.

Essa relação é utilizada para que se garanta a integridade dos dados armazenados no banco de dados de autenticação, permitindo que o sistema utilize um banco de dados mais robusto e modular, não armazenando os dados localmente. Para acessar os dados, é estabelecido uma padronização dos dados transmitidos, através de um gerenciador de SQL no lado servidor.

Esse gerenciador é responsável por receber as requisições de *logins* e as validar, além de fornecer as credenciais e níveis de acesso de cada usuário registrado.

Figura 19: Fluxograma relacional entre o Modelo e o gerenciador do servidor remoto.



Fonte: Próprio autor.

A fim de garantir uma segurança e integridade nos dados transitados entre o sistema e o banco de dados, fora aplicado um método de criptografia baseado em chaves simétricas, que criptógrafa todos os pacotes transmitidos. A criptografia com chaves simétricas é um método de criptografia que utiliza a mesma chave para tanto a criptografia quanto a descriptografia dos dados.

Nesse tipo de criptografia, a mesma chave é usada tanto pelo remetente quanto pelo destinatário para garantir a confidencialidade dos dados. O remetente aplica a chave compartilhada aos dados originais para transformá-los em uma forma ilegível, conhecida como texto cifrado. Em seguida, o texto cifrado é transmitido de forma segura ao destinatário. O destinatário utiliza a mesma chave para aplicar a operação inversa, ou seja, descriptografar o texto cifrado e obter os dados originais.

A principal vantagem da criptografia com chaves simétricas é a sua velocidade e eficiência, pois o processo de criptografia e descriptografia é relativamente rápido. No entanto, a segurança desse método depende da proteção adequada da chave compartilhada. Se a chave for comprometida, a confidencialidade dos dados pode ser comprometida.

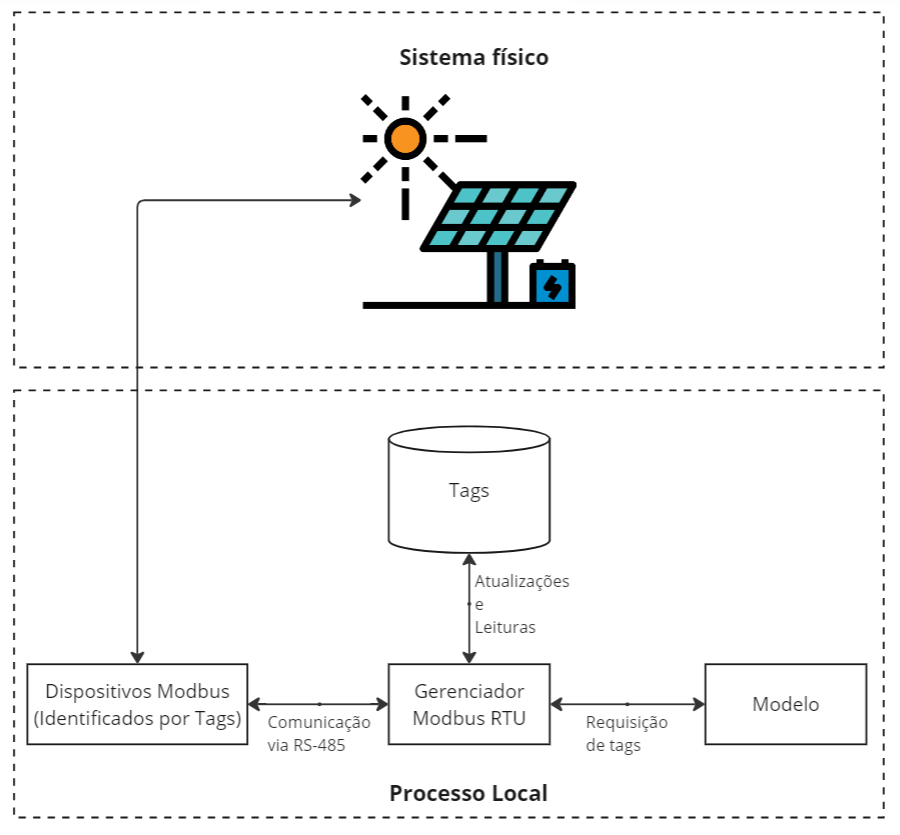
De forma simplificada, o sistema supervisório se conecta ao servidor através do envio de uma chave de criptografia que será a chave comum aos dois. Como forma de resposta, o servidor envia uma mensagem padrão de recebimento utilizando a chave de criptografia gerada pela interface. Se o lado cliente, usando a mesma chave de criptografia, consegue descriptografar a mensagem padrão, significa então que cliente e servidor estão conectados e sincronizados entre si.

Para realizar o *login*, o Modelo recebe as credenciais do sistema através do controlador e os envia criptografados para o gerenciador SQL hospedado no lado servidor, se as credenciais estiverem de acordo, então o servidor retorna as mensagens contendo as informações do usuário, tal qual, nome, nível de acesso, fotos, registros e etc.

#### Interface Modbus RTU

O supervisório se comunica com o *Tracker* utilizando o barramento serial RS-485, conectando-se a conversor serial que permite que o computador comunique com esse barramento. O tráfego de mensagens nessa interface é gerenciado pelo Modelo e o fluxograma de processos pode ser visto na Figura 20.

Figura 20: Fluxograma entre Modelo e interface serial RS-485.



Fonte: Próprio autor.

É possível perceber que a comunicação passa por um banco de dados que possui a responsabilidade de armazenar as comunicações realizadas entre o sistema e o supervisório. Esse banco de dados armazena as informações de registradores *Modbus* em uma tabela organizada por sistema, tipo de registrador e tipo de dados. Um exemplo dos dados armazenados nas tabelas descritas pode ser visto na Tabela 2.

Tabela 2: Exemplo de tabela de *tags* Modbus

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Endereço do sistema** | **Nome** | **Endereço do registrador** | **Tamanho** | **Tipo** | **Valor** | **Última atualização** |
| 0x12 | estado | 0x00 | 1 | INT | 1 | 2023-07-04 10:22:02 |
| 0x12 | alarme | 0x01 | 1 | INT | 0 | 2023-07-04 10:22:02 |
| 0x12 | azimute | 0x03 | 2 | FLOAT | 23,45 | 2023-07-04 10:22:02 |
| 0x12 | zenite | 0x05 | 2 | FLOAT | 127,54 | 2023-07-04 10:22:02 |
| 0x12 | Pos. 1 | 0x07 | 2 | FLOAT | 23,46 | 2023-07-04 10:22:02 |
| 0x12 | Pos. 2 | 0x09 | 2 | FLOAT | 127,51 | 2023-07-04 10:22:02 |
| ... | ... | ... | ... | ... | ... | .... |

Fonte: Próprio autor

Dessa forma, é possível se armazenar os últimos estados do sistema e se preciso, realizar rotinas de geração de relatórios.

### Tela gráfica

A parte de visualização do sistema e iteração entre operador e sistema é de responsabilidade da Tela de visão, que gerencia todos os desenhos dos objetos na tela e atrela os valores de *tags* em determinados objetos.

A interface gráfica possui 6 telas únicas que juntas, elas realizam as renderizações de todo sistema supervisório. São elas:

1. Tela de *Login*;
2. Tela de Inicio;
3. Tela de Mapa;
4. Tela de conexão Serial;
5. Tela de Diagnósticos.

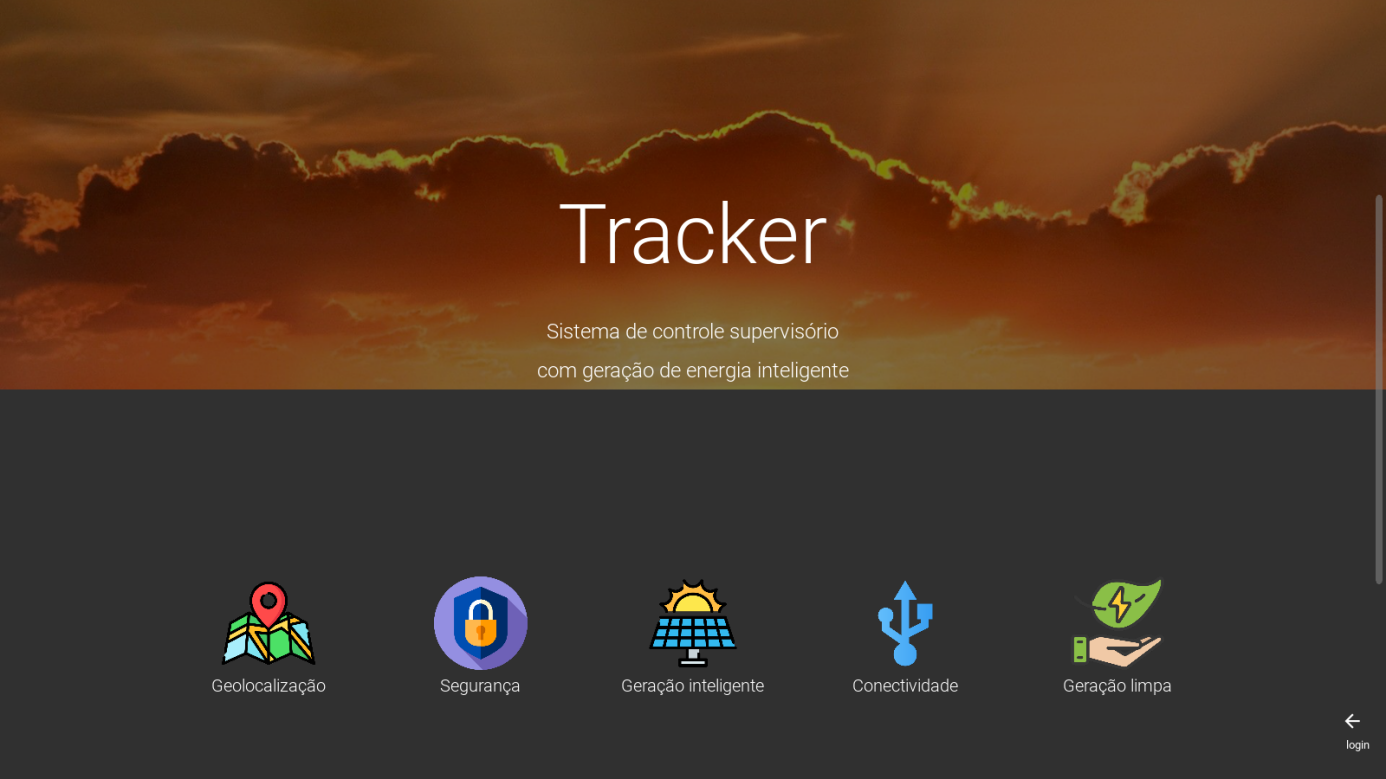
Essas telas serão descritas com detalhes na seção 5.

# INTERFACE DE CONTROLE

## Tela de Login

A primeira tela a ser implementada, é a tela de *login*, onde é efetuado a identificação do usuário e além disso, é a primeira tela a ser acessada pelo usuário do sistema, portanto, ela deve possuir uma boa impressão ao entrar. Na Figura 21 é possível se visualizar como a tela de *login* se parece ao ser acessada pela primeira vez.

Figura 21: Tela de *login*.

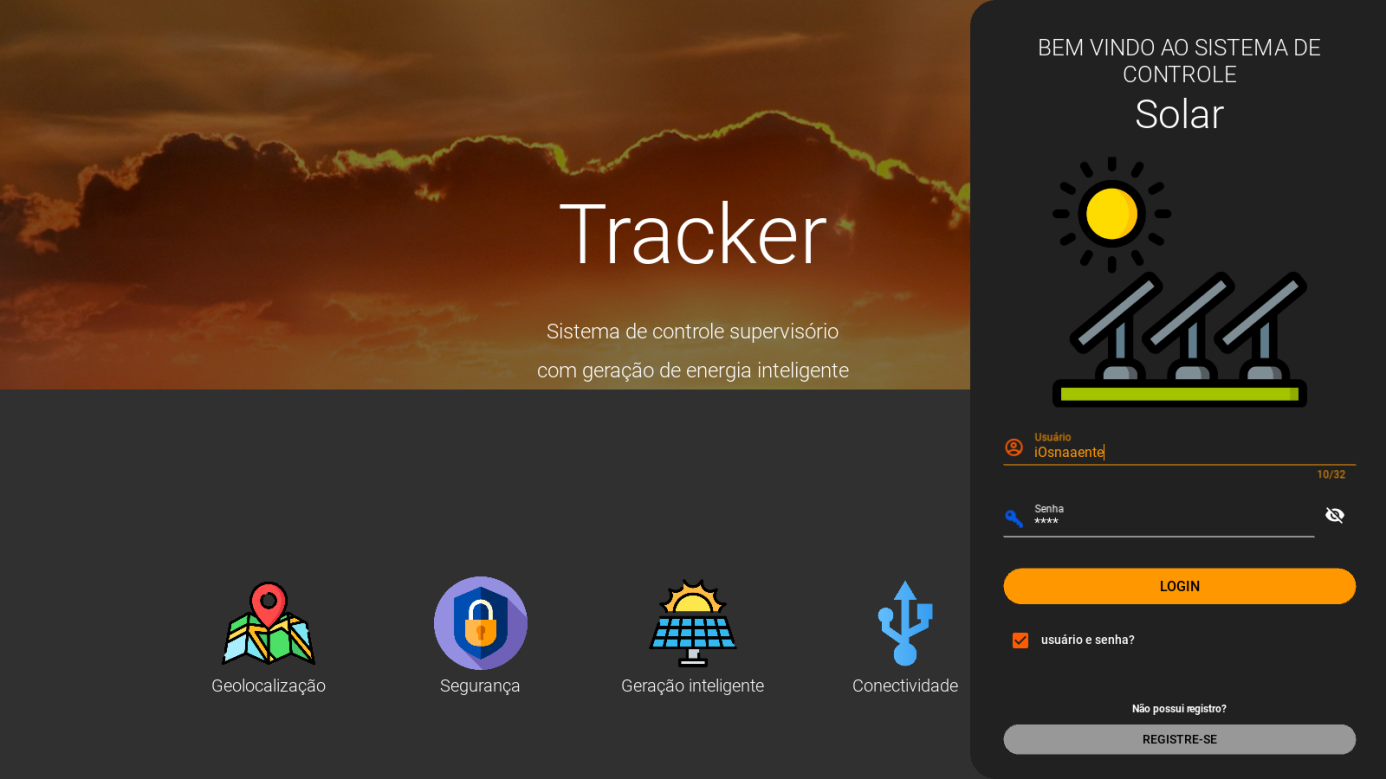


Fonte: Próprio autor.

Esta tela deve ser capaz de transmitir a sensação de que se trata de um sistema de rastreamento solar e passar uma visão dos benefícios de se utilizar esse sistema. A tela possui uma imagem de fundo que remete ao poder do sol e é possível ver os ícones informativos na parte inferior, que abordam os benefícios do sistema.

Para fins de clareza e com o intuito de não poluir a tela com muitas informações, a secção de *login* pode ser acessada clicando no botão localizado na borda inferior direita ou arrastando a barra lateral direita da direita para a esquerda. A barra de *logins* será acessada como mostrado na Figura 22.

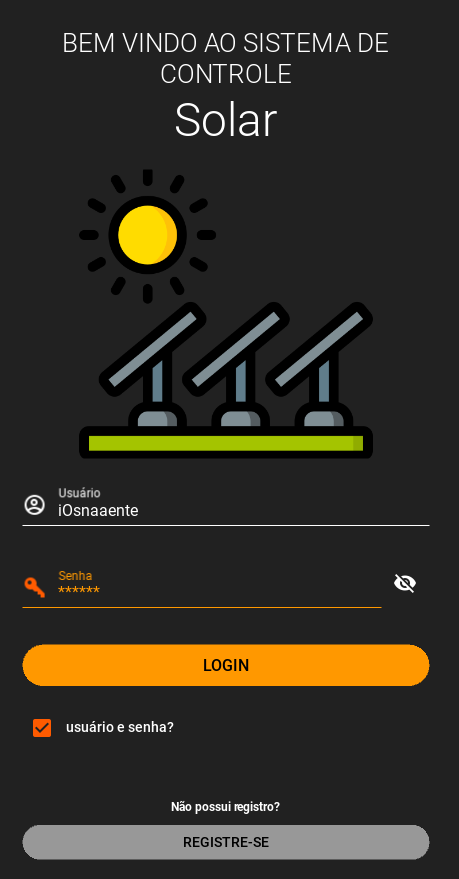
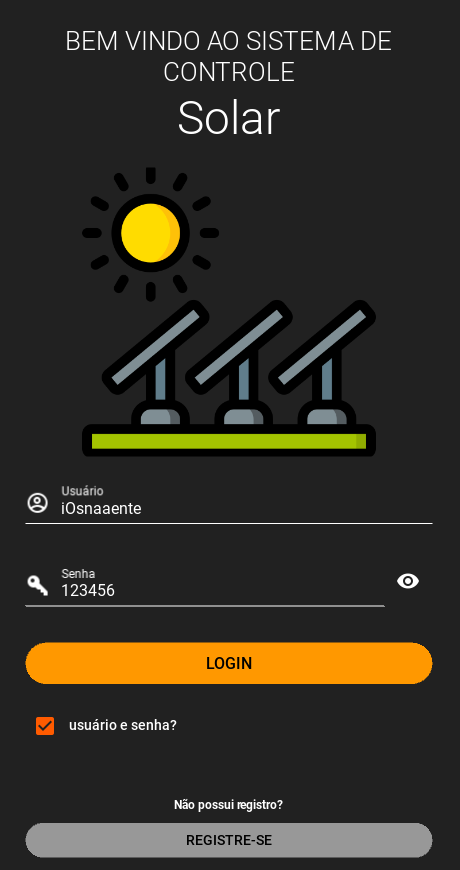
Figura 22: Acesso ao menu de login.



Fonte: Próprio autor.

A secção de *login* possui como finalidade a identificação do usuário do sistema, ao qual deverá dar como entrada as credencias de usuário e senha de acesso. Além disso, a fim de manter o *input* mais intuitivo, ele possui sinalizações de campo ativo e permite que a senha seja exposta se necessário (vide Figura 23).

Figura 23: Seção para login.

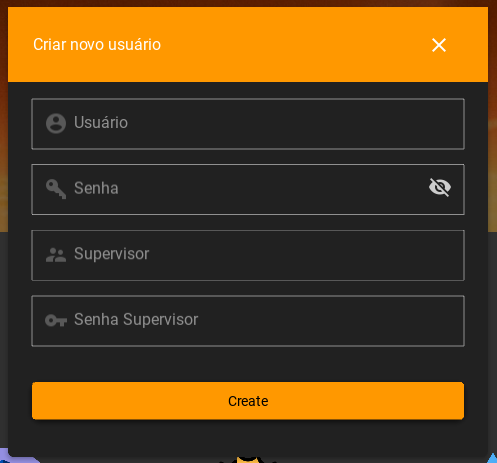
Fonte: Próprio autor.

É possível também, se criar um novo usuário, clicando no botão “REGISTRE-SE” o qual será aberto uma pequena janela como mostrado na Figura 24. Nessa janela é possível se ver 4 campos, sendo eles:

1. Usuário: Nome do usuário que será registrado;
2. Senha: Senha do usuário que será registrado;
3. Supervisor: Nome do supervisor ou administrador do sistema;
4. Senha do supervisor: Senha do administrador ou supervisor do sistema.

Esse método de registro permite que seja seguro criar um novo usuário, somente com a permissão de um supervisor ou administrador do sistema, garantindo a segurança e integridade do *Tracker*.

Figura 24: Tela de registro de novo usuário.



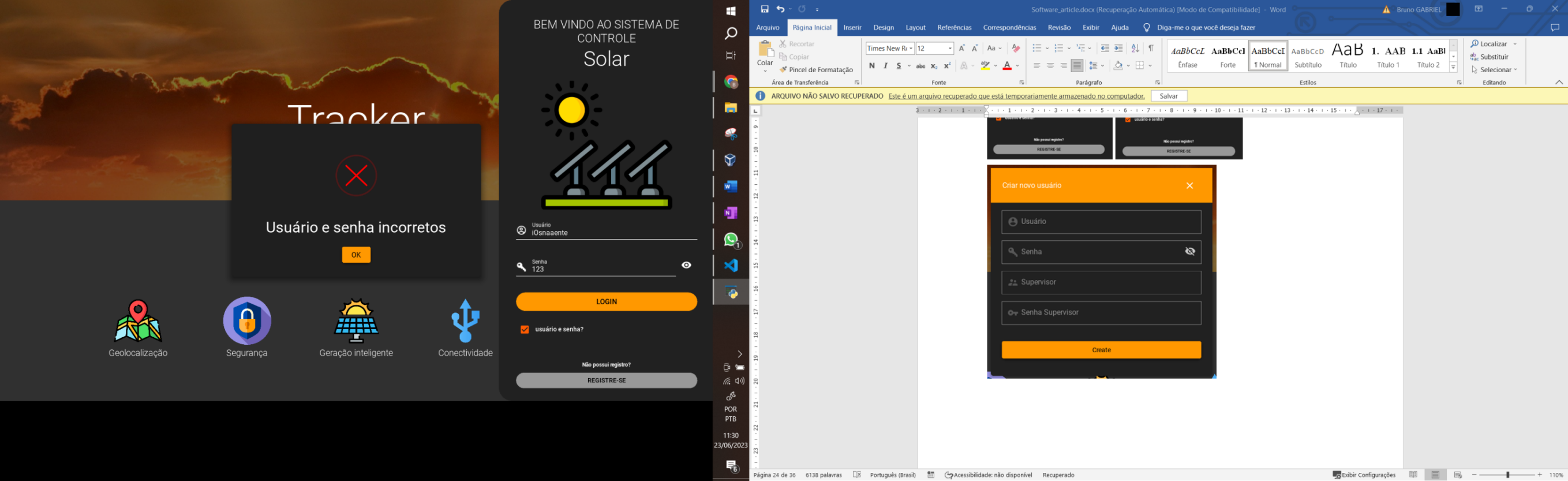
Fonte: Próprio autor.

Outra vantagem do sistema de *login* é que ele é capaz de lançar mensagens de aviso para erros de autenticação, como:

1. Criação de usuários repetidos;
2. Usuário e senha incorretos;
3. Falta de conexão com o banco de dados.

Na Figura 25 é possível se ver uma mensagem de erro de autenticação ativa.

Figura 25: Mensagem de erro de autenticação.



Fonte: Próprio autor.

Se autenticado com sucesso, a tela de *login* dará espaço para a nova tela de inicio que começará a ser renderizada assim que aberta.

## Tela de inicio

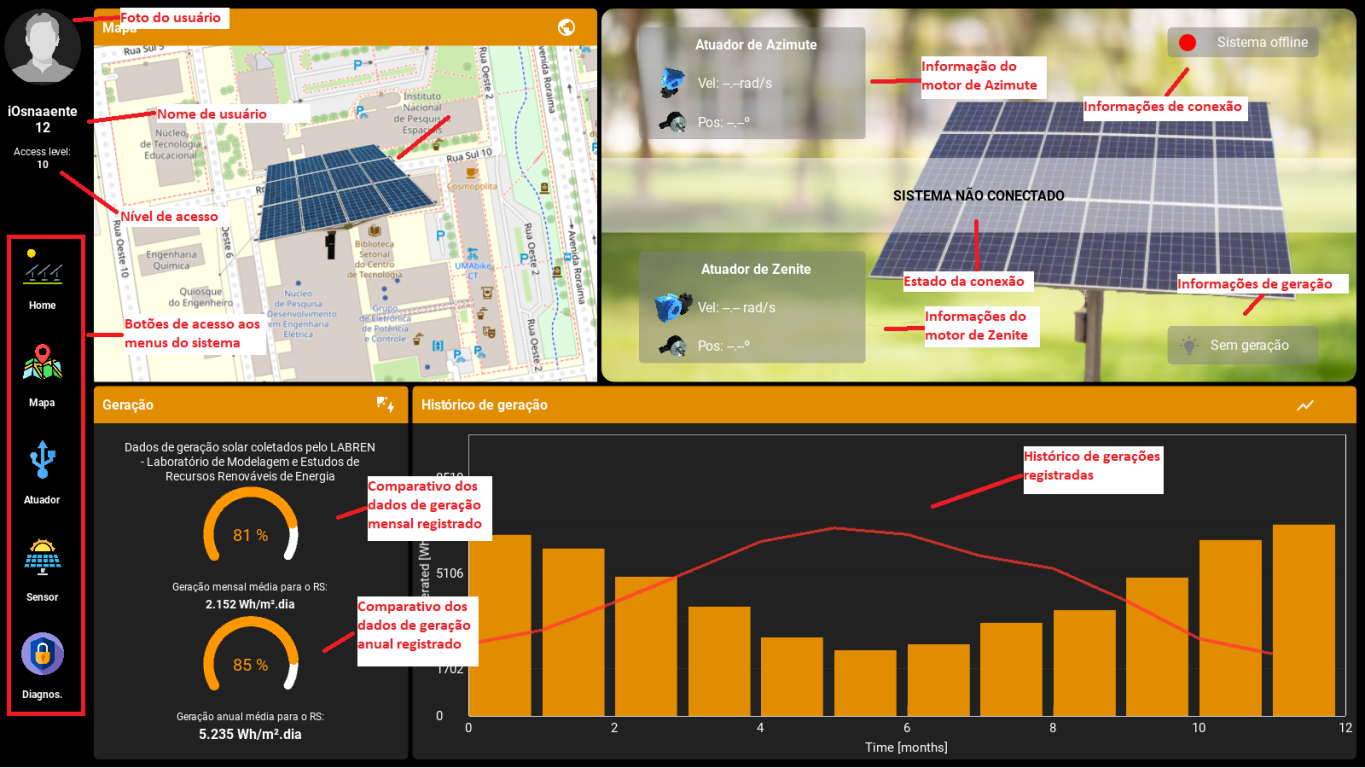
A primeira tela ao se acessar o *software* de supervisão, será a tela de inicio, a qual possui a responsabilidade de passar um panorama geral sobre o sistema, desde informações de conexão com o sistema a valores de geração solar médio registrado no seu banco de dados. Na Figura 26 é possível se observar a disposição das janelas informativas da tela inicial e na Figura 27 é possível se identificar as funções de cada objeto presente na tela inicial.

Figura 26: Visualização da tela inicial.



Fonte: Próprio autor.

Figura 27: Descrição dos objetos presentes na tela inicial.

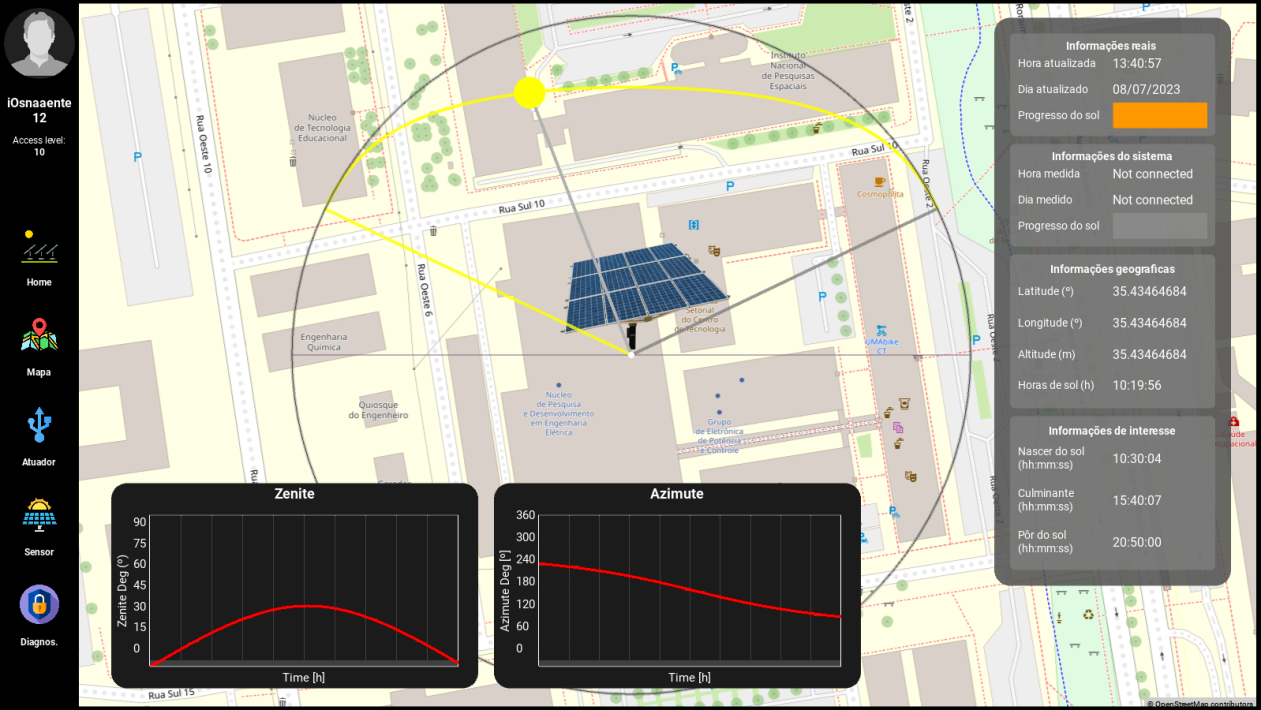


Fonte: Próprio autor.

## Tela de mapas

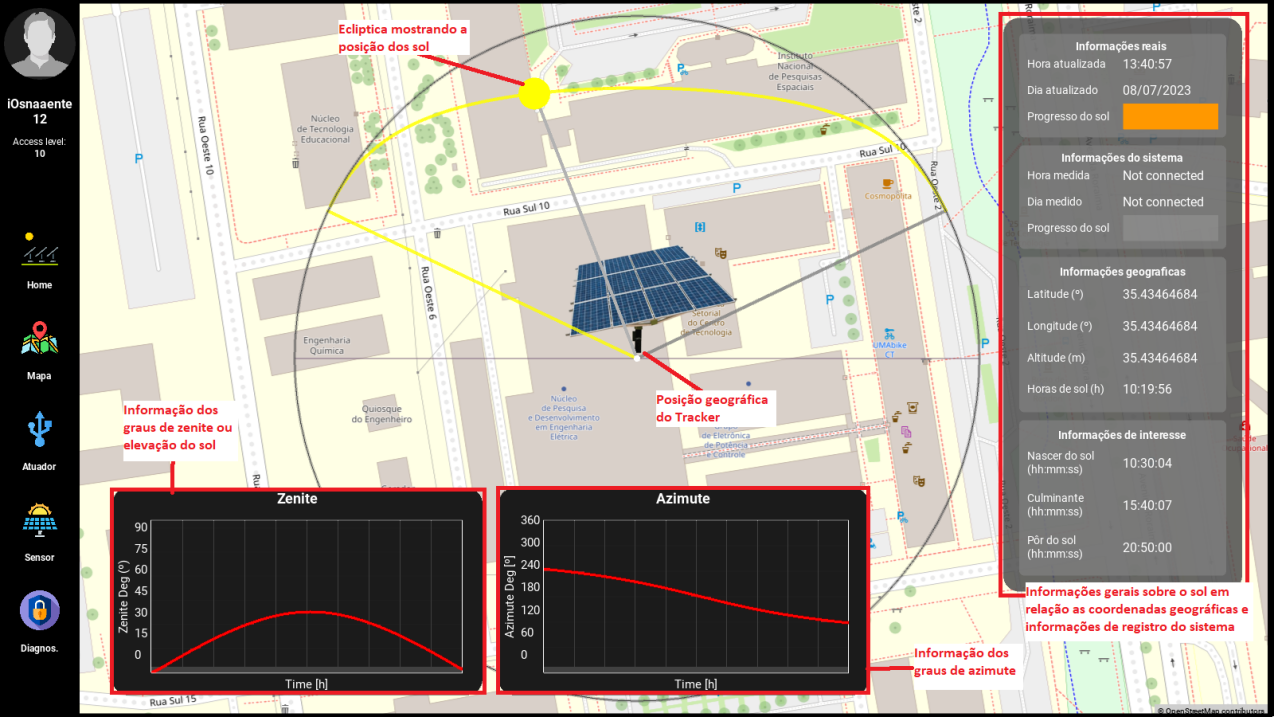
Na navegação do sistema, utilizando o botão do menu chamado “Mapa”, é possível entrar na tela de visualização do sistema em um mapa. Essa tela é responsável por mostrar visualmente a posição geográfica dos sistemas de rastreadores cadastrados, além mostrar informações a respeito do comportamento do sol. Na Figura 28 é possível visualizar a tela de mapas e na Figura 29 os detalhes de cada objeto em tela.

Figura 28: Visualização da tela de mapas do sistema.



Fonte: Próprio autor.

Figura 29: Visualização detalhada da tela de mapas do sistema



Fonte: Próprio autor

## Tela de conexão serial

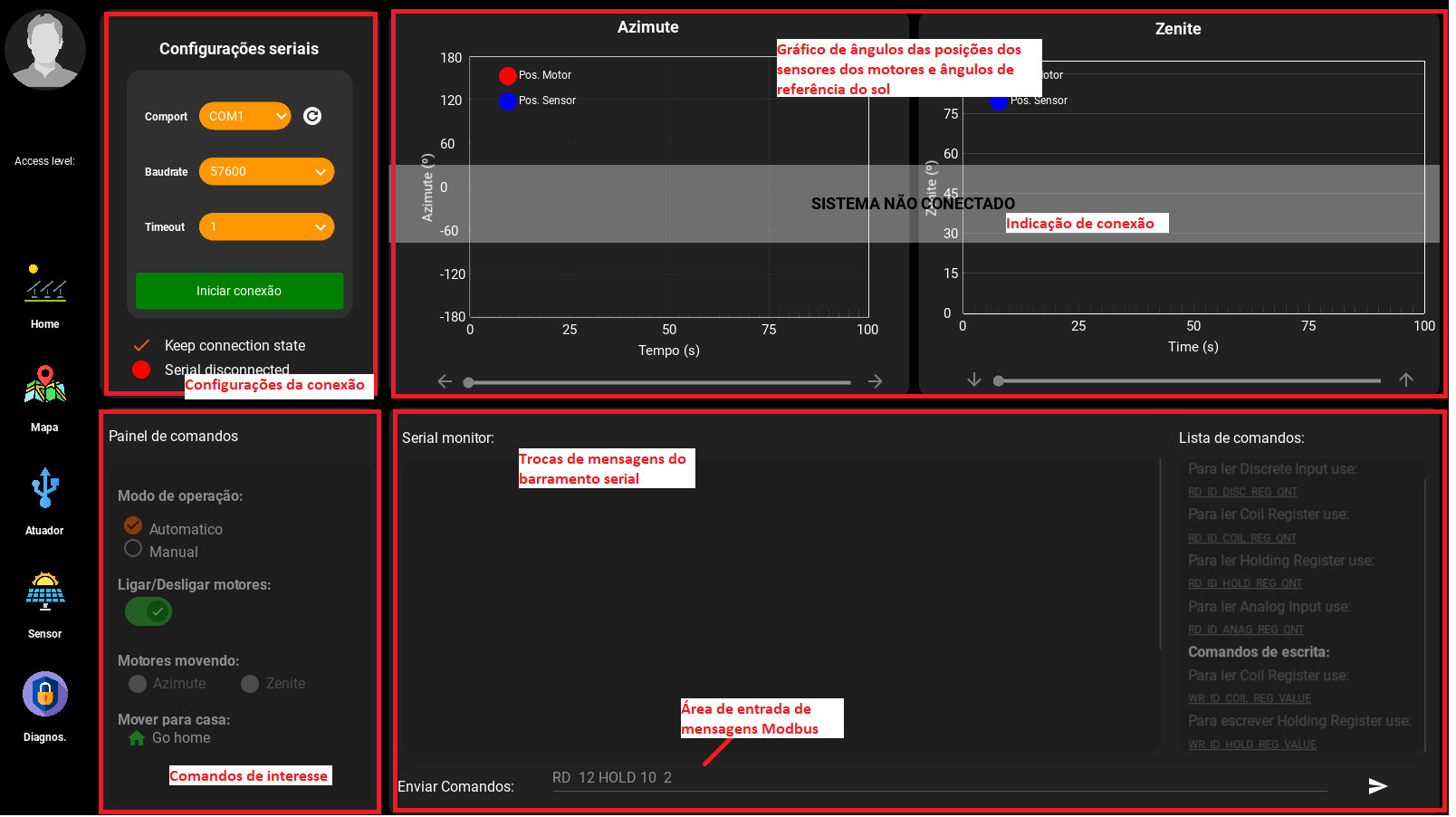
Na tela de conexão serial, é apresentado as principais funções de conexão e controle do sistema supervisionado. Na Figura 30 é mostrado a visão geral do sistema com um sistema desconectado. Nessa janela é possível se realizar a configuração da conexão serial como: selecionar a porta serial a ser utilizada pelo sistema, selecionar a velocidade da conexão e selecionar o prazo de resposta (*timeout*) dos escravos *Modbus* (vide Figura 31).

Figura 30: Visualização da tela de conexão serial.



Fonte: Próprio autor.

Figura 31: Visualização detalhada da tela de conexão serial.



Fonte: Próprio autor.

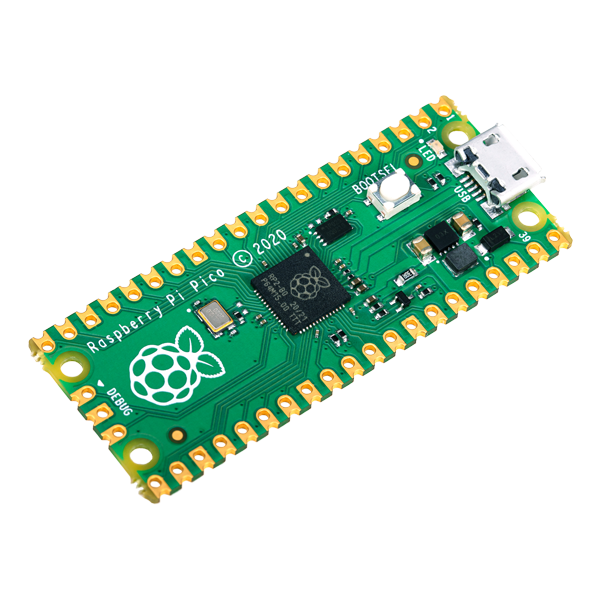
# SISTEMA físico

Com o intuito de se testar o sistema desenvolvido, fora utilizado um dispositivo microcontrolado que fosse capaz de simular um sistema de rastreamento solar de dois eixos e que realiza-se a interface *Modbus* para comunicação com o supervisório.

## Microcontrolador Raspberry Pi Pico

Como sistema de controle embarcado no sistema Tracker, fora utilizado um Raspberry Pi Pico, um microcontrolador da família Raspeberry Pi (vide Figura 32).

Figura 32: Visão geral do Raspberry Pi Pico



Fonte: Disponível em: <https://s3-sa-east-1.amazonaws.com/robocore-lojavirtual/1313/pinout.jpg>

O microcontrolador possui embarcado o chip RP2040, um microcontrolador *Dual-core Cortex M0*+ com 256kb de memória RAM e 2MB de memória *flash*. Ele possui 26 pinos de GPIO (sigla de *General Purpose Input/Output*) permitindo 2 conexões SPI (sigla de *Serial Peripheral Interface*), 2 conexões I2C (sigla de *Inter-Integrated Circuit*), 2 conexões UART (sigla de *Universal Asynchronous Receiver/Transmitter*), possui 3 canais com conversores analógico/digital de 12 *bits* e 16 canais PWM (sigla do inglês *Pulse Width Modulation*) programáveis (vide Anexo 1). A principal vantagem do microcontrolador esta no fato de que ele pode ser programado usando a linguagem de programação C ou então *Python*.

## Motores de corrente continua (DC)

Para a visualização do sistema em funcionamento, fora escolhido 2 motores de corrente contínua de baixa rotação e alto torque, a fim de simular os movimentos de azimute (*Yaw*) e zênite do sistema (*Pitch* ou *Roll*). Os motores utilizados podem ser vistos na Figura 33.

Figura 33: Motor DC escovado utilizado no sistema.

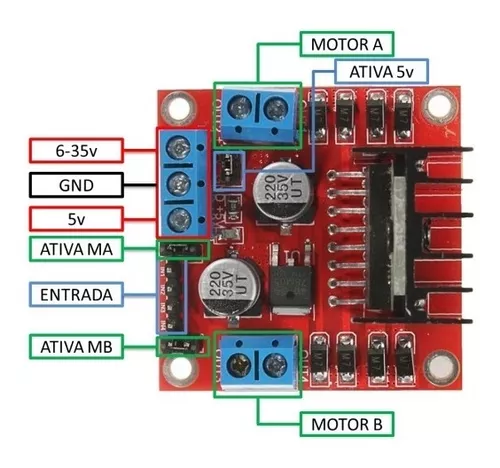


Fonte: Disponível em <https://www.amazon.com.br/redu%C3%A7%C3%A3o-resfriado-m%C3%A1quina-ferramenta-grava%C3%A7%C3%A3o-substitu%C3%ADvel/dp/B0BMZX3GQB>.

## Driver de acionamento de motores DC

Para se realizar o acionamento dos motores DC, fora utilizado um *driver* de motor DC também conhecido como ponte-H. O modelo de ponte-H utilizado foi o L298N (vide Figura 34) que possui dois canais de controle de motores independentes com tensão de operação de 6 a 35V@2A por canal. Para se realizar o controle dos motores, é utilizado 2 GPIOs para controle de direção e 1 GPIO para controle de velocidade por canal.

Figura 34: Ponte-H L298N de dois canais.



Fonte: Disponível em: <https://http2.mlstatic.com/D_NQ_NP_718297-MLB32623382003_102019-O.webp>

## Sensores de posição angular magnéticos

A fim de se realizar a medição da rotação dos motores para se realizar um controle em malha fechada do sistema, se utilizou um sensor magnético do modelo AS5600 (vide Figura 35-a). Esse sensor permite realizar a medição da posição de um eixo sem fazer contato com ele, utilizando um imã de polaridade diametral (vide Figura 35-b). Esse sensor possui uma resolução de 12 *bits*, sendo capaz de medir 4096 valores entre 0 e 360º, sentindo variações de 0,087890625º.

O sensor utiliza a interface de comunicação I2C com o microcontrolador, sem a possibilidade de se trocar o endereço I2C, dessa forma são necessários, duas interfaces I2C para comunicar com os dois sensores.

Figura 35: Sensor AS5600 (a) Visão geral do sensor (b) Imã com polaridade diametral

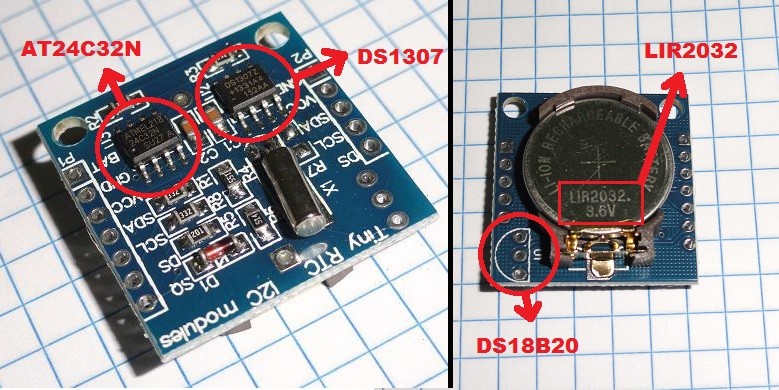
|  |  |
| --- | --- |
| IMG_256 |  |
| **(a)** | **(b)** |

Fonte: Disponível em: <https://www.fermarc.com/as5600-sensor-de-angulo-magnetico-potenciometro>

## Relógio de tempo real

Para se manter a hora do sistema atualizada mesmo com o microcontrolador desligado, foi previsto um sensor RTC (sigla de *Real Time Clock*) para armazenar e contar os dias e horas de maneira precisa, uma vez que os dados temporais são fundamentais para manter o sistema com elevada eficiência de geração. O módulo RTC utilizado foi o DS1307 fabricado pela Maxim-C, embarcado em um placa juntamente com um *chip* EEPROM (sigla de *Electrically-Erasable Programmable Read-Only Memory* ) de marcação AT24C32N, capaz de realizar até 1 milhão de gravações e armazenar os dados por até 100 anos sem perdas (vide Figura 36). Ambos *chips* (RTC e EEPROM) possui comunicação via interface I2C. A placa possui também uma pilha recarregável do modelo LIR2032.

Figura 36: Visão geral da placa com RTC embarcado.



Fonte: Disponível em <https://embarcados.com.br/modulo-tiny-rtc-i2c-parte-1/>

## Dispositivos de desarme de alimentação

A fim de realizar o arme e desarme do sistema, fora utilizado um relé RAS-1210 (vide Figura 37) com capacidade de acionar cargas de até 24V@10A através de uma bobina de 12VDC.

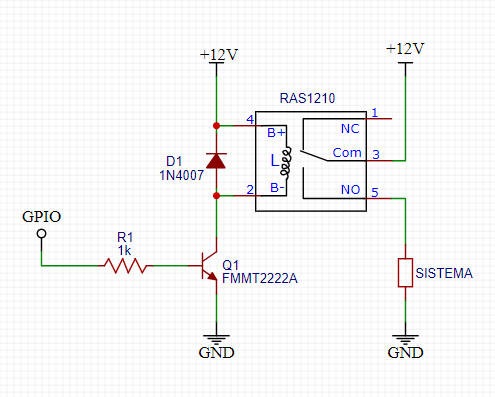
Figura 37: Relé de acionamento 12V/24V@10A RAS-1210



Fonte: Disponível em: <https://www.dayoutlet.com.br/seminovos/ras-1210-mini-rele-10a-12vdc-pack-4pcs>

Para acionamento do relé fora utilizado um transistor NPN, FMMT2222A do tipo SMD (sigla de *Surface Mounted Device*) com marcação 1P, como chave e um diodo 1N4007 para roda livre como segue o esquemático da Figura 38. A ativação do relé se dá pela saída de uma GPIO vindo do microcontrolador.

Figura 38: Esquemático de acionamento do relé RAS-1210



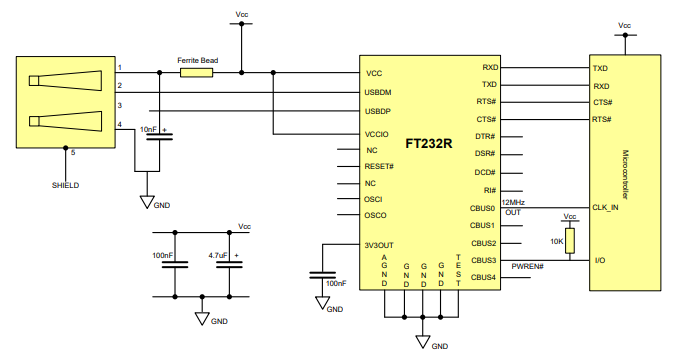
Fonte: Próprio autor

## Interface USB-UART

Como o sistema é dotado de uma comunicação *Modbus* RTU, utilizando uma comunicação serial entre o computador e microcontrolador embarcado, é necessário que seja feito uma conversão do padrão USB (sigla de *Universal Serial Bus*) para o padrão UART (sigla e *Universal Asynchronous Receiver/Transmitter*) a fim de estabelecer uma conexão reconhecível entre os sistemas.

O módulo conversor utilizado foi genérico, possuindo o componente integrado FT232R na configuração de interface USB para UART dedicado a microcontroladores, como segue na Figura 39 retirada da folha de dados do componente.

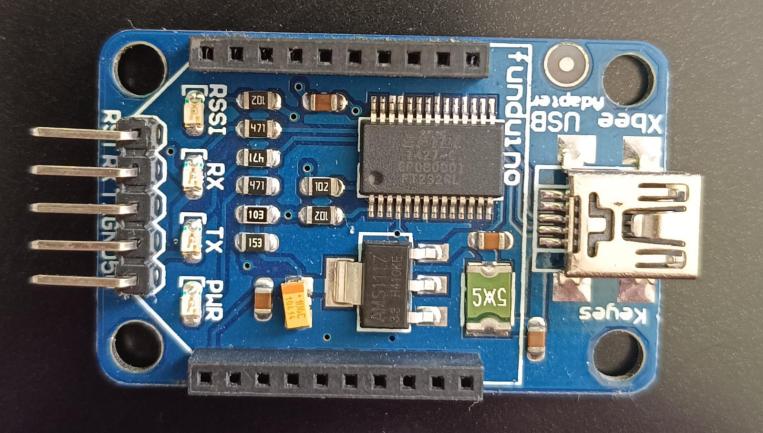
Figura 39: Configuração do FT232R para conversor USB para UART



Fonte: Folha de dados do componente FT232R. Disponível em <https://ftdichip.com/wp-content/uploads/2020/08/DS\_FT232R.pdf> Acesso em: 15 de out. de 2023.

Na Figura 40 pode ser visto o módulo da *funduino* utilizado como conversor USB para UART que faz o uso do componente integrado FT232R como conversor dedicado.

Figura 40: Módulo funduino utilizando o FT232R como conversor.



Fonte: Próprio autor.

## Fonte de alimentação

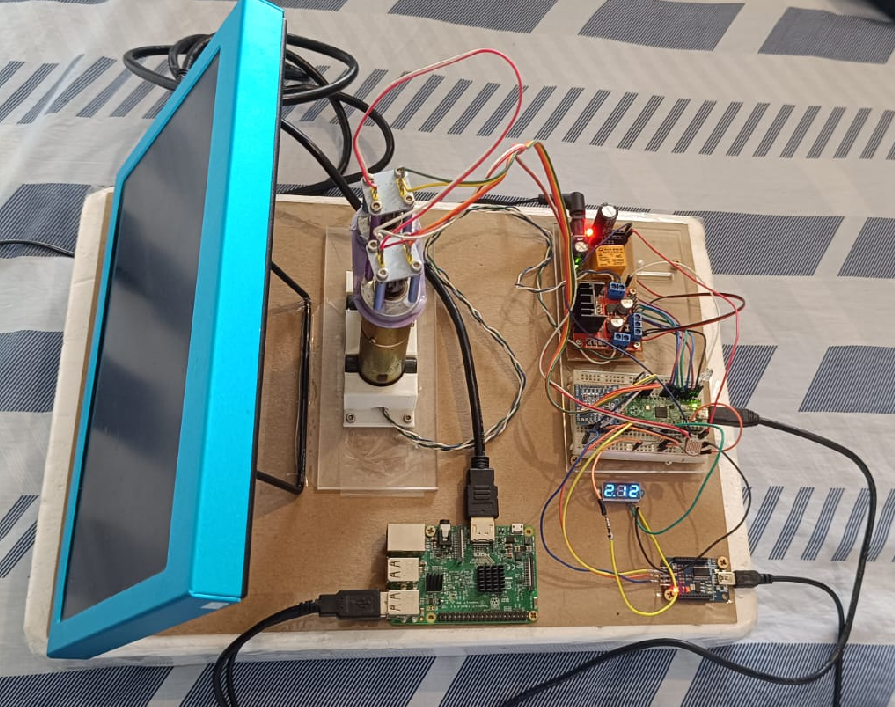
Como fonte de alimentação, fora utilizado uma fonte de alimentação 220Vac@1A com frequência de operação de 50 a 60Hz para 12Vdc@2,5A. Essa fonte alimenta os motores de corrente contínua atualizados no sistema e também é regulada para 5V para alimentar o circuito lógico da ponte H usada para controlá-los. Além disso, a tensão de 5V é utilizada para alimentar o microcontrolador Raspberry Pi Pico que possui um conversor *Buck* embarcado que rebaixa essa tensão para 3,3V de forma regulada, sendo através dessa fonte de alimentação que os módulos sensores são alimentados, a fim de manterem o nível lógico compatível aos 3,3V do microcontrolador.

O sistema é capaz de ser alimentado com uma tensão de até 30Vdc, sendo essa tensão limitada pelo circuito da ponte H utilizado.

## Sistema completo

O sistema de teste completo pode ser visto na Figura 41.

Figura 41: Foto do sistema completo.



Fonte: Próprio autor.

Como esse sistema possui a finalidade de validar o *software* da interface de controle do rastreador solar, não houveram preocupações em seu funcionamento pleno.

# RESULTADOS

Na seção, serão apresentados os resultados obtidos pelo sistema de supervisão desenvolvido, sendo os pontos de interesse avaliados, a confiabilidade do sistema de autenticação do sistema de *login*, a comunicação do sistema físico com o supervisório e o funcionamento do sistema de forma remota, utilizando o supervisório como controlador do sistema de rastreio.

## Autenticação de usuário

A parte fundamental para garantir a segurança de qualquer processo esta na garantia de que usuários não habilitados, utilizem o sistema. Dessa forma, a primeira análise do sistema como um todos, será direcionada à questão da autenticação dos usuários, avaliando o sistema de credenciamento de novos usuários e o sistema de autenticação de *logins* no sistema.

Se tratando de uma aplicação com servidor remoto, os pacotes transmitidos entre a aplicação e o servidor, podem ser capturados usando um *sniffer* de pacotes TCP/IP.

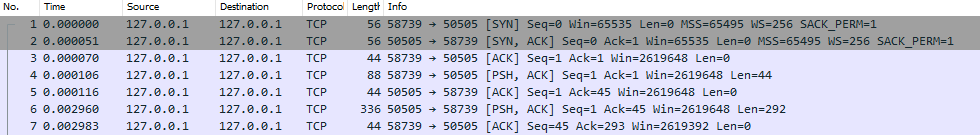
Um *sniffer* é um *software* ou *hardware* que permite que se monitore o tráfego de pacotes de dados transmitidos e recebidos. O *sniffer* utilizado para a captura dos pacotes TCP/IP trafegados foi o *Wireshark*, um *software* de código aberto disponível para os principais sistemas operacionais. Para realizar essa analise, a versão utilizada do Wireshark foi a 4.0.10 disponível em <https://www.wireshark.org/> com acesso em 15 de out. de 2023.

### Registro e usuários

Para a criação de um novo usuário é necessário que o supervisório esteja na página inicial, com o servidor ativo, para que a haja uma conexão TCP/IP estabelecida entre o sistema e o servidor que gerencia as credenciais. Utilizando o Wireshark para monitorar as conexões TCP/IP é possível perceber a conexão sendo estabelecida assim que o sistema é ativo, analisando o padrão de mensagens TCP/IP pelo característico *Three-Way Handshake* (aperto de mão triplo) como mostrado nos três primeiro pacotes da Figura 42.

O aperto e mão triplo é uma forma que o protocolo TCP/IP utiliza para estabelecer uma conexão entre dois sistemas a fim de garantir que ambos sistemas estejam conectados e sincronizados. Esse sistema foi utilizado visando garantir uma maior segurança entre a troca de mensagens, uma vez que toda troca de dados entre os sistemas é garantida através de um pacote de confirmação de recebimento, assinalada com um campo especial dentro do quadro TCP chamado ACK, como mostrado na Figura 42.

Figura 42: Troca de mensagens TCP/IP entre sistema e servidor.



Fonte: Próprio autor via *Wireshark.*

Uma vez estabelecido uma conexão TCP/IP com sucesso o usuário já estará apto para registrar um novo usuário no sistema, para isso basta entrar no campo de registro de novo usuário e preencher os dados requisitados corretamente.

Como o sistema utiliza uma criptografia simétrica entre a troca de informações, mesmo analisando os pacotes transmitidos com o *Wireshark*, não é possível se identificar qualquer informação da transação, uma vez que a criptografia embaralha os valores e cria uma sequência única de *bytes* transmitidos, cujo só é possível se identificar o teor das mensagens utilizando a mesma chave simétrica. Essa chave simétrica é única e gerada para cada conexão TCP/IP estabelecida no sistema.

Dessa forma, é garantido que as informações sensíveis do sistema serão trafegadas de forma segura. A fim de exemplificar esse método de criptografia, foi registrado um usuário fictício a fim de se analisar as informações trafegadas. Na Figura 43 é possível visualizar em (a) as credenciais utilizadas no teste de criação de novo usuário e (b) as informações simplificadas recebidas pelo *backend* do servidor.

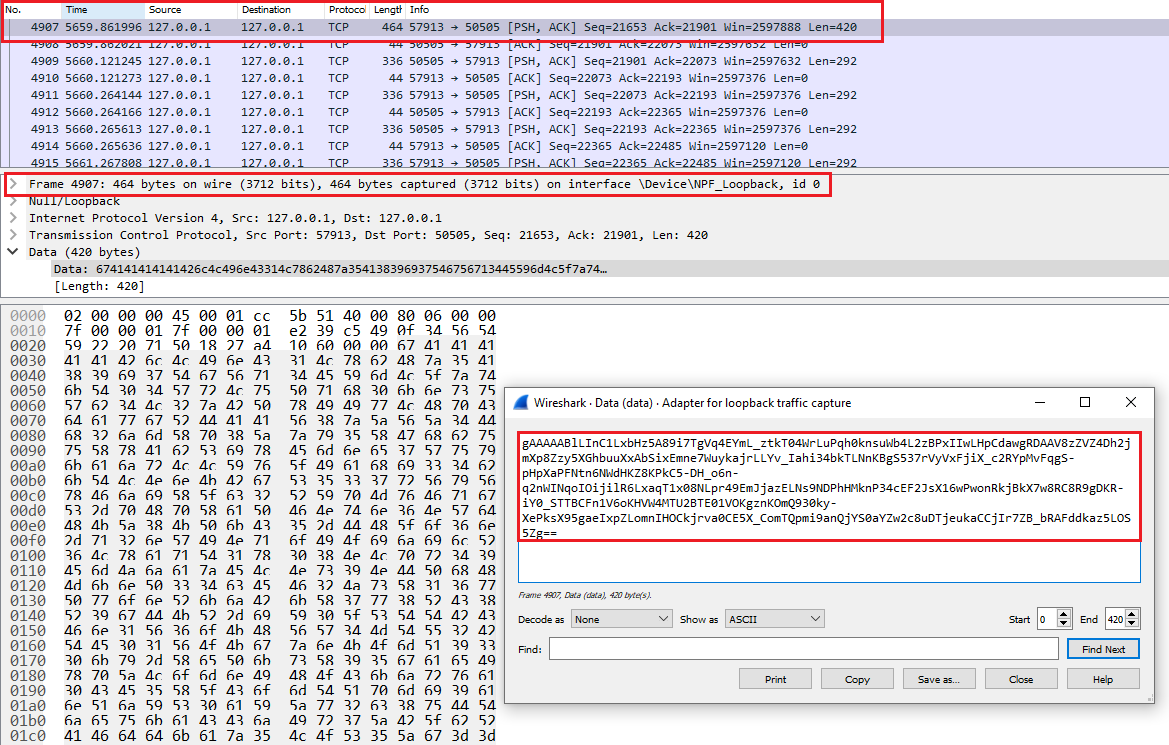
Percebe-se que os valores trafegados fazem parte de um dicionário em python, com 5 campos de dados onde cada dado é mostrado de forma literal, ou seja, em caracteres ASCII. Na Figura 44 é possível se identificar o pacote da mensagem trafegada e analisando o campo de dados úteis, se percebe que os dados trafegados são aparentemente aleatórios e em uma quantidade de bytes superior ao valor de caracteres apresentados, por motivo da criptografia.

Figura 43: Informações trafegadas para criar novo usuário (a) Lado supervisório (b) Lado servidor.

|  |  |
| --- | --- |
|  | Recebido: {  'type': 'NEW\_USER',  'username': 'Teste',  'password': 'Registro',  'manager\_group': 'sup',  'manager\_psd': 'sup123'  }  Resposta b'NEW USER CREATED' |
| (a) | (b) |

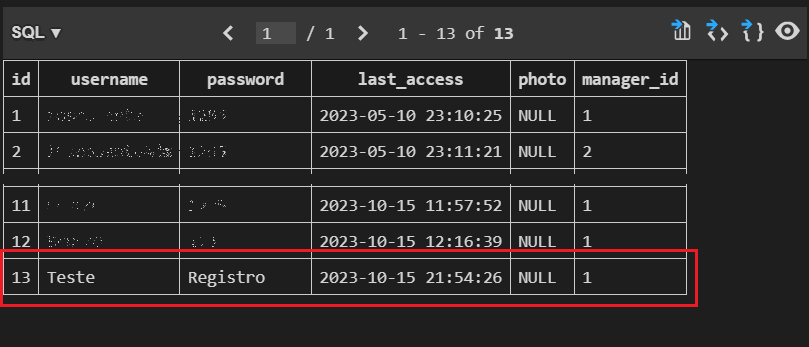
Fonte: Próprio autor.

Figura 44: Mensagem para criação de novo usuário.

Fonte: Próprio autor.

Avaliando o *backend* do servidor é possível ver que o usuário foi criado com sucesso como mostrado na Figura 45. O lado servidor não é acessado por nenhuma aplicação externa ao supervisório, portanto o *backend* pode ser visualizado com as permissões de super usuário sem problemas.

Figura 45: Registro de usuário no banco de dados do servidor.

Fonte: Próprio autor - Banco de dados editado para proteger dados sensíveis.

Sendo assim, o registro de usuários fora feito de forma segura, garantindo que os dados transmitidos não fossem facilmente capturados por outros *softwares* farejadores (*sniffers*) garantindo a integridade dos dados trafegados. Dessa forma, a interface de controle pode contar com bancos de dados externos, hospedados em diferente regiões geográficas desde que o sistema possua acesso a Internet, trazendo uma maior flexibilidade a região onde o sistema será instalado.

### Autenticação de login

A autenticação de *login* ou entrada de usuários ao sistema é e fundamental importância, uma vez que o sistema deve garantir que somente usuários devidamente registrado no banco e dados do sistema tenha acesso ao sistema com os devidos níveis de acesso do sistema.

O sistema é dividido entre dois tipos principais de usuários, sendo os usuários supervisores e os usuários administrados, onde o primeiro goza de menos acesso do que o segundo, não sendo possível acessar regiões sensíveis do sistema como o controle manual do sistema físico por exemplo.

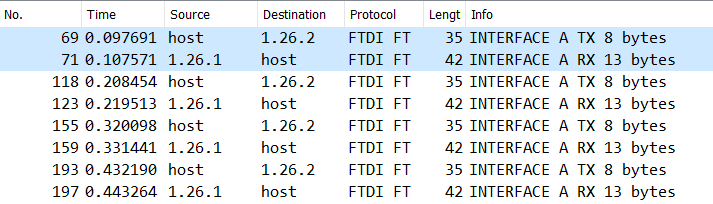
## Comunicação com o sistema

O sistema supervisório se conecta com o sistema de controle físico através de uma interface USB/UART, passando por um conversor serial do tipo FTDI (do inglês *Future Technology Devices International*) que é responsável pela conversão dos dados da USB para o meio físico correto. Quando a interface é iniciada e o sistema esta dentro da tela inicial, acontece uma rotina de verificação pelos dispositivos USB conectados no computador e se o sistema alguma vez já fora conectado, ele retém as credenciais, dessa forma, acontece um sincronismo entre a interface e o sistema de controle.

Se o sistema supervisório identifica uma conexão com o sistema físico, é iniciada a troca de mensagens entre eles utilizando o protocolo *Modbus RTU*. Como na tela inicial são apresentados as animações de posicionamento do sistema, ele tenta fazer a leitura dos valores dos sensores de posição de azimute e zênite, realizando essa medição em um intervalo de tempo de pelo menos 100ms. Para realizar essa leitura, o sistema solicita ao sistema Modbus conectado, através do envio de uma mensagem, que os dois registradores de posição sejam lidos. A tabela de endereços dos registradores *Modbus* é conhecida e esta disponível em Apêndice 1.

Na Figura 46 é possível se observar essa troca de mensagens arbitrária entre o supervisório e o sistema.

Figura 46: Captura dos pacotes transitados pela USB via *Wireshark USBCap*.



Fonte: Próprio autor.

Como esperado, é identificado um padrão de troca de mensagens onde, inicialmente o sistema supervisório, identificado como Host, envia 8 *bytes* para o destino 1.26.2 via padrão FTDI. A mensagem *Modbus* esta encapsulada dentro do frame USB que contém 35 *bytes*, onde esta possui um *payload* de 8 *bytes* que corresponde ao *frame* *Modbus*.

A fim de validar o *frame Modbus*, pode-se analisar o *payload* do padrão USB e verificar a estrutura enviada. Na Figura 47 pode-se ver o conteúdo do *payload* da troca de mensagens 69 destacado em azul.

Figura 47: Conteúdo do frame USB.



Fonte: Próprio autor.

Pode-se validar que foram enviados 8 *bytes* como pacote, com uma estrutura que se assemelha a de leitura de múltiplos registradores do tipo *INPUT REGISTERS* como manda o protocolo *Modbus*. A estrutura é mostrada de forma mais clara pela Tabela 3.

Tabela 3: Identificação do *Payload* do frame USB transmitido.

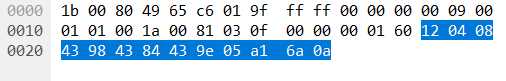
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID do dispositivo** | **Código de função** | **Endereço do primeiro registrador** | **Quantidade de registradores** | **Checagem de erro** |
| 0x12 | 0x04 | 0x00 0x00 | 0x00 0x04 | 0xF3 0x6A |
| 18 | 4 | 0 | 4 | 62.314 |

Fonte: Próprio autor.

Sabendo que o código de função de leitura de registradores do tipo de entradas analógicas é o código 4 e reconhecendo o padrão da estrutura, pode-se confirmar que a mensagem transmitida esta de acordo com o protocolo *Modbus*.

Como esperado, uma vez que o mestre solicita uma leitura dos registradores 0x00 e 0x02, o dispositivo deverá responder a mensagem com 13 *bytes*, que correspondem respetivamente a: 1 *byte* de ID do dispositivo, 1 *byte* com o código de função, 1 *byte* com o número de *bytes* dos registradores que será transmitido, 8 *bytes* de dados, uma vez que foram solicitados a leitura de 4 registradores onde cada registrador possui 2 *bytes*, por ser registradores de 32 *bits* e por fim mais 2 *bytes* para a checagem de erro. Dessa forma totalizando 13 *bytes*, como visto na Figura 46. Já na Figura 48 pode ser visto o conteúdo do pacote recebido do sistema.

Figura 48: Pacote de dados recebido do sistema físico como resposta.

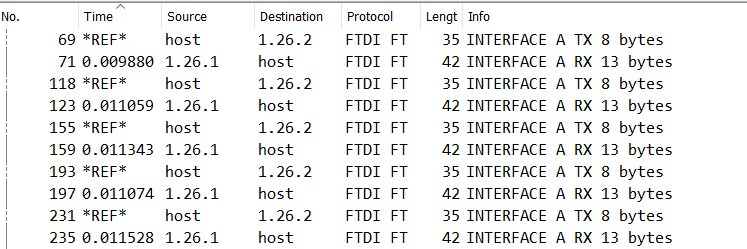


Fonte: Próprio autor.

Como esperado, foram recebidos 13 *bytes* na sequência descrita. Como os valores dos registradores são do tipo de 32 *bits* com codificação IEEE-754 do tipo de ponto flutuante, os valores contidos em cada registrador não podem ser identificados de foram trivial.

Por fim, pode-se então verificar o tempo de resposta entre cada solicitação e resposta para se verifica quanto tempo é levado desde a solicitação dos valores até sua respectiva resposta por parte do controlador. Na Figura 49, foram usados os mesmos pacotes da Figura 46, no entanto foram referenciados os tempos de solicitação de mensagem com o tempo zero, para se comparar a diferença de tempo para se receber a resposta.

Figura 49: Medição do tempo de resposta do dispositivo Modbus.

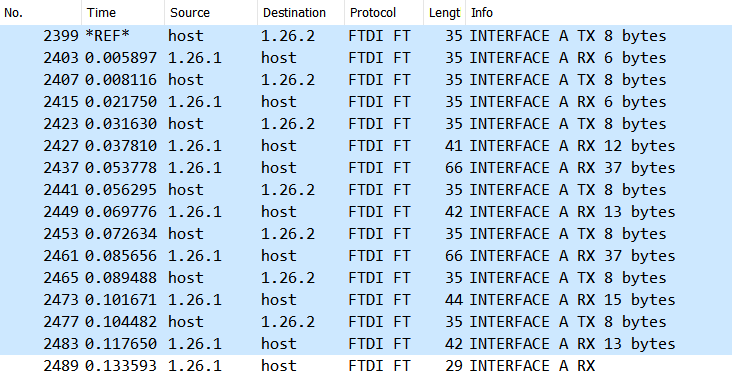


Fonte: Próprio autor.

Percebe-se um tempo médio de resposta de 10.97ms que sai do mínimo de 9.88ms para o máximo de 11.52ms durante as 5 medições consideradas. Dessa forma, pode-se confirmar que o sistema é capaz de realizar as medições em um ciclo de pelo menos 100ms como esperado inicialmente, podendo também ser atualizado em 60 vezes por segundo juntamente com a parte gráfica do sistema supervisório que trabalha com um tempo de renderização de 16ms.

Além das medições de posição do sistema, o supervisório também possui uma rotina de leitura de todos os registradores do sistema de forma cíclica com período de 1 segundo. Na Figura 50 é possível ver as 7 transações necessárias para se realizar a leitura de todos os registradores do sistema físico.

Figura 50: Mensagens trocadas para a leitura de todos registradores Modbus do sistema.



Fonte: Próprio autor.

Infelizmente, para a leitura de todos os registradores do sistema, é necessário 117,65ms para que o supervisório solicite, receba e processe todos os pacotes, o que reduz o tempo de ciclo das medições drasticamente. Todavia, para a aplicação em questão, não é necessário se realizar a medição de todos registradores simultaneamente, sendo assim, o sistema supervisório poderá operar de forma fluida, ou então, se o tempo de resposta não for primordial, a medição de todos os ciclos do *Modbus* podem ser mantidas, ou então, realizadas de uma forma mais inteligente, de forma que o tempo de leitura de todos os registradores seja feito de forma distribuída, não travando o sistema pelos 117,65ms, mas sim em tempos menores de forma mais recorrente.

Dessa forma, o sistema em questão se mostrou capaz de transmitir pacotes rápido o suficiente para se ter uma interface com atualizações dos valores eficiente, com aspectos fluidos, sem os travamentos por falta ou taxa de atualização, no entanto, nas condições apresentadas não é possível se obter um carácter determinístico para se efetuar um controle efetivo na planta. Na Figura 51 é possível se observar os gráficos da janela serial com os motores em operação, sendo possível perceber a fluidez da medição do sistema.

Figura 51: Visualização da funcionalidade dos gráficos de posição do sistema.

Foto da tela serial com os sensores em funcionamento

Fonte: Próprio autor.

# Considerações finais

A implementação do sistema foi bem-sucedida, evidenciando a eficácia da autenticação encriptografada. A capacidade de conexão tanto a servidores locais quanto externos destaca a versatilidade da interface supervisória. Adicionalmente, a interface é capaz de verificar a disponibilidade do servidor de *login* em tempo de execução. Se o servidor não estiver acessível, uma mensagem de aviso é exibida na tela, garantindo a integridade de uso dos usuários finais, evitando que falhas desconhecidas possam surgir.

O sucesso do sistema de *login*, especialmente com autenticação encriptografada, não apenas assegura o acesso seguro ao sistema, mas também reflete a preocupação com a proteção de dados sensíveis, uma vez que mascaram as credenciais dos usuários do sistema. A implementação eficaz dessas medidas de segurança fortalece a credibilidade e aceitação do sistema, especialmente em ambientes onde a proteção de informações é prioritária.

Esses elementos não apenas garantem um acesso seguro e confiável ao sistema, mas também estabelecem um padrão elevado de segurança e funcionalidade. Assim, a implementação bem-sucedida dessas medidas confirma a viabilidade e aceitação da interface supervisória em contextos nos quais a segurança da informação é uma prioridade crucial.

Em suma, o sistema de login se mostrou eficaz e com um banco de dados rodando no *backend* como servidor, que armazena as credenciais de *login*, que pode ser utilizado local e remotamente sem problemas. Com a criptografia dos dados de *login*, o sistema se mostrou seguro para ser utilizado na rede. As rotinas de criação e registros de usuários ocorreu de forma satisfatória, possuindo avisos de cadastro, senhas incorretas e indicações de servidor não localizado ou inacessível, se mostrando eficaz para o propósito pelo qual foi desenvolvido.

A interface gráfica do sistema desempenha um papel fundamental na experiência do usuário, e os resultados obtidos indicam que ela atendeu eficazmente a esse propósito. A disposição dos elementos visuais fora disposta de formas intuitivas, buscando transmitir com clareza a apresentação das informações e suas funções. A fluidez do sistema, conforme evidenciada nos testes, revelou-se consistente, proporcionando uma resposta rápida às interações do usuário e com a disposição de animações, que trazer uma boa experiência de uso pelos usuários.

Houve um cuidado pelo aspecto visual da interface, sendo desenhado para possuir uma boa impressão. Quando o sistema é incido, animações são iniciadas para dar uma boa impressão aos usuários, além disso, há uma harmonia na escolha das cores do sistema, que tornam o processo de utilização mais intuitivos e agradáveis. As transições de páginas da interface ocorrem de maneira suave e são eficientes.

Boa parte da responsabilidade pela fluidez do sistema esta que ele esta baseado em programação concorrente, sendo utilizados *Threads* de funcionamento para responsabilidades gráficas e para comunicação, paralelizando as operações. Isso garante que a interface visual não seja afetada pelos tempos de comunicação entre interface e sistema. A preocupação com a memória compartilhada entre as *threads* foi feita de forma eficiente, não tendo ocorrido conflitos de acesso a memórias compartilhadas o que poderia acarretar em quebras da operação do sistema.

Abordagens de armazenamento das informações coletadas pelo supervisório em bancos de dados como *sqlite* não foram utilizadas na versão final. Para o armazenamento dos dados em bancos de dados, a velocidade das rotinas de leitura do sistema e escrita foram prejudicadas de forma que a fluidez na leitura dos dados do sistema perdessem fluidez e prejudicassem a taxa de atualização da interface gráfica. Esse problema poderá ser corrigido em pesquisas para aplicações futuras.

Um dos pontos mais sensíveis da aplicação esta na comunicação com o sistema físico através da interface de comunicação *Modbus*. Por ser realizada através da interface USB, o sistema necessita utilizar drivers de comunicação FTDI e gerenciar essas comunicações. O sistema é capaz de escolher em quais portas USB o sistema estará conectado, todavia, não é capaz de garantir que o sistema conectado esteja apto a receber e responder as mensagens que o sistema espera, sendo um ponto crítico para o funcionamento de toda interface, que mesmo que esteja mostrando as mensagens de “sistema conectado”, ele pode não estar.

Todavia, se a interface estiver rodando em um sistema dedicado, esse problema tende a ser minimizado a ponto de não ser relevante. Esses pontos descritos podem ser aperfeiçoados e são incentivados que sub-rotinas de verificação sejam adotadas no sistema, para garantir maior integridade ao sistema.

Para a comunicação com o sistema físico apresentado, a velocidade nas trocas de mensagens simples (posição dos sensores e medição de energia dos painéis) entre o sistema e a interface foram feitas em tempos médios de 11ms. Para a troca de mensagens com um volume maior de *bytes*, foram necessários em média 117,65ms. Dessa forma, pode-se concluir que a comunicação ocorreu de forma dinâmica e rápida o suficiente para manter a fluidez em pontos críticos da interface, como na visualização de gráficos suave com boa taxa de atualização. No entanto, para a realização de um controle com determinismo temporal, ainda se faz necessário adaptações no sistema e é visto como uma melhoria futura.

Em conclusão, o desenvolvimento e implementação do sistema supervisório atingiu êxito e foi concluído de forma satisfatória. Os resultados obtidos foram positivos na eficiência do sistema de *login*, na integração bem-sucedida com o sistema físico via protocolo *Modbus*, e na robustez da interface gráfica evidenciam não apenas a conclusão do projeto, mas também seu desempenho consistente e confiável. A fluidez operacional da interface, aliada à resposta ágil do sistema, proporcionaram uma boa experiência de uso da interface. Este trabalho não apenas representa a conclusão bem-sucedida de um projeto significativo, mas também destaca o potencial do sistema para aplicações práticas e sua contribuição para avanços na área de supervisão de sistemas de energia renovável.

A interface de controle dos sistemas de rastreamento solar apesar de estarem em uma versão estável e funcional, ainda estão pendentes de muitas funções auxiliares que podem trazer novas funcionalidades importantes para a operação do sistema, como:

1. Implementação de uma rotina de *scan* dos endereços *Modbus* do sistema físico, evitando que seja necessário se alterar manualmente cada endereço e função dos registradores *Modbus*.
2. Rotina de verificação de dispositivos conectados via USB com compatibilidade com o sistema, garantindo correto funcionamento do sistema com os dispositivos conectados.
3. Rotina de multiprocessamento para controle e monitoramento de multiplos dispositivos Tracker na mesma interface.
4. Comunicação distribuída de leitura dos registradores *Modbus* do sistema físico.
5. Implementação de leitura e gravação dos registradores em banco de dados para geração de relatórios.

# REFERÊNCIAS

ASSIS, Daniel Paulo. **INTERFACE GRÁFICA COMO FATOR DETERMINANTE NO DESENVOLVIMENTO DE SOFTWARE**. Instituto Municipal de Ensino Superior de Assis - IMESA e a Fundação Educacional do Munícipio de Assis – FEMA, 2012.

COPÉRNICO. N. **Commentariolus**: PequenoComentário de Nicolau Copérnico sobre suas próprias hipóteses acerca dos movimentos celestes. Tradução por Roberto de Andrade Martins. 2. ed. São Paulo: Livraria da Física, 2003

GREEN POWER: **Parque solar São Gonçalo**. 2021. Disponível em: < <https://www.enelgreenpower.com/pt/nossos-projetos/highlights/parque-solar-sao-goncalo> >. Acesso em: 01 set 2021.

JUNIOR, Ervaldo Garcia. **Introdução a Sistemas de Supervisão, Controle e Aquisição de Dados: SCADA**. Alta Books Editora, 2019.

SANTOS, Mayara Helena Moreira Nogueira dos. **Sistema de monitoramento integrado baseado nos novos procedimentos de rede do operador nacional do Sistema (ONS) para adequação a subestações digitais**.

sieBRASIL: **Ministério de minas e energia do brasil – Sistema de Informação de Energias SIE -** **Capacidade Instalada de Geração Elétrica**: Disponível em: <<https://www.mme.gov.br/SIEBRASIL/consultas/reporte-dato42-jerarquizado.aspx?oc=30181&or=30182&ss=2&v=1>>. Acesso em: 01 set 2021.

SILVA, Rutelly Marques da. Energia Solar no Brasil:  **dos incentivos aos desafios. 2015**. Disponível em: <<http://www12.senado.gov.br/publicacoes/estudos-legislativos/tipos-de-estudos/textos-para-discussao/td166>>. Acesso em: 01 set. 2021

RICHARDSON, L: **ENERGYSAGE**. mai 2018, Disponível em: < <https://news.energysage.com/the-history-and-invention-of-solar-panel-technology/> >. Acesso em: 01 set 2021.

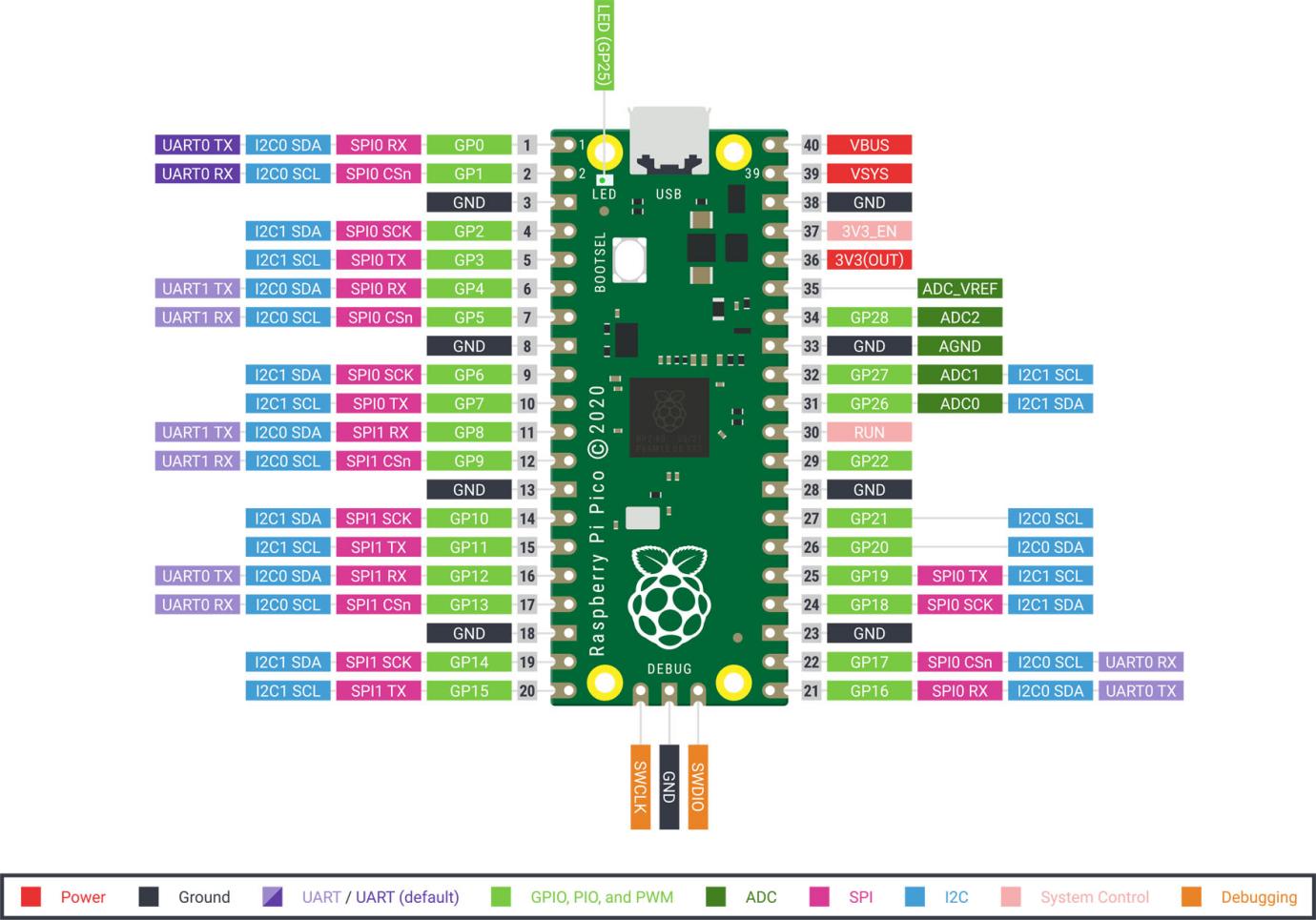
TANENBAUM, A. S. – **Redes de Computadores** – 5ª Ed., Editora ‎ Pearson Universidades, 2011.

TOSSATO, Claudemir Roque; MARICONDA, Pablo Rubén: **O método da astronomia segundo Kepler.** Scientiae Studia, v. 8, p. 339-366, 2010.

VALLDOREIX GREENPOWER: **The Benefits of Solar Trackers.** Julho de 2015.Disponível em: < <http://www.valldoreix-gp.com/the-benefits-of-solar-trackers/> >. Acesso em: 01 set 2021.

# ANEXO

Anexo I: Configuração de pinos do Raspberry Pi Pico.



Fonte: Disponível em: <https://s3-sa-east-1.amazonaws.com/robocore-lojavirtual/1313/pinout.jpg>

# APÊNDICE

Apêndice I: Tabela de endereços Modbus usados no sistema físico para testes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **REGISTRADOR DO TIPO INPUT** | | | **REGISTRADOR DO TIPO HOLDING** | | |
| **DESCRIÇÃO** | **ENDEREÇO** | **TIPO** | **DESCRIÇÃO** | **ENDEREÇO** | **TIPO** |
| INPUT\_POS\_GIR | 0x00 | FLOAT | HR\_PV\_GIR | 0x00 | FLOAT |
| INPUT\_POS\_ELE | 0x02 | FLOAT | HR\_KP\_GIR | 0x02 | FLOAT |
| INPUT\_AZIMUTE | 0x04 | FLOAT | HR\_KI\_GIR | 0x04 | FLOAT |
| INPUT\_ZENITE | 0x06 | FLOAT | HR\_KD\_GIR | 0x06 | FLOAT |
| INPUT\_GENERATION | 0x08 | FLOAT | HR\_AZIMUTE | 0x08 | FLOAT |
| INPUT\_TEMP | 0x0A | FLOAT | HR\_PV\_ELE | 0x0A | FLOAT |
| INPUT\_PRESURE | 0x0C | FLOAT | HR\_KP\_ELE | 0x0C | FLOAT |
| INPUT\_SENS\_CONF\_GIR | 0x0E | FLOAT | HR\_KI\_ELE | 0x0E | FLOAT |
| INPUT\_SENS\_CONF\_ELE | 0x10 | FLOAT | HR\_KD\_ELE | 0x10 | FLOAT |
| INPUT\_YEAR | 0x12 | INT | HR\_ALTITUDE | 0x12 | FLOAT |
| INPUT\_MONTH | 0x13 | INT | HR\_LATITUDE | 0x14 | FLOAT |
| INPUT\_DAY | 0x14 | INT | HR\_LONGITUDE | 0x16 | FLOAT |
| INPUT\_HOUR | 0x15 | INT | HR\_STATE | 0x18 | INT |
| INPUT\_MINUTE | 0x16 | INT | HR\_YEAR | 0x19 | INT |
| INPUT\_SECOND | 0x17 | INT | HR\_MONTH | 0x1A | INT |
|  |  |  | HR\_DAY | 0x1B | INT |
|  |  |  | HR\_HOUR | 0x1C | INT |
|  |  |  | HR\_MINUTE | 0x1D | INT |
|  |  |  | HR\_SECOND | 0x1E | INT |
| **REGISTRADOR DO TIPO COIL** | | | **REGISTRADOR DO TIPO DISCRETE** | | |
| **DESCRIÇÃO** | **ENDEREÇO** | **TIPO** | **DESCRIÇÃO** | **ENDEREÇO** | **TIPO** |
| COIL\_POWER | 0x00 | BIT | DISCRETE\_FAIL | 0x00 | BIT |
| COIL\_LED | 0x01 | BIT | DISCRETE\_POWER | 0x01 | BIT |
| COIL\_M\_GIR | 0x02 | BIT | DISCRETE\_TIME | 0x02 | BIT |
| COIL\_M\_ELE | 0x03 | BIT | DISCRETE\_GPS | 0x03 | BIT |
| COIL\_LEDR | 0x04 | BIT |  |  |  |
| COIL\_LEDG | 0x05 | BIT |  |  |  |
| COIL\_LEDB | 0x06 | BIT |  |  |  |
| COIL\_SYNC\_DATE | 0x07 | BIT |  |  |  |

Fonte: Próprio autor.