UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LEONARDO LAURYEL BATISTA DOS SANTOS

SPIDER: Uma infraestrutura para programação e coleta de dados de FPGAs remotamente

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Profª. Drª. Fernanda Gusmão de Lima Kastensmidt

Porto Alegre
2023

“A educação é a arma mais poderosa
que você pode usar para mudar o mundo”
— NELSON MANDELA

AGRADECIMENTOS

Gostaria de registrar aqui o meu agradecimento a diversas pessoas que foram fundamentais durante essa jornada.

Primeiramente, agradeço à minha família por sempre me incentivarem a buscar conhecimento e jamais medir esforços para que eu pudesse ter acesso a uma educação de qualidade. A vocês, Acelmo, Leonilda, Laysla e Anderson, muito obrigado por estarem sempre ao meu lado, ofertando amor, apoio e compreensão. Sem vocês, nada disso teria sido possível.

A todos os meus amigos que estiveram sempre ao meu lado, apoiando-me, dando conselhos e tornando essa caminhada mais divertida. Gostaria de fazer um agradecimento especial à Andréa Santos e à Fernanda Bonetti por estarem constantemente ao meu lado, dando dicas e me motivando durante o desenvolvimento deste trabalho.

À minha orientadora deste trabalho de conclusão de curso, Prof[a]. Dr[a]. Fernanda Gusmão de Lima Kastensmidt por todo o apoio, por me guiar e pelas conversas divertidas. Ao Dr. Fabio Benevenuti pela proatividade e por estar sempre disposto a me ajudar e sanar minhas dúvidas.

Aos meus orientadores do projeto que participei durante a graduação, Prof. Dr. Luciano Paschoal Gaspary e Prof. Dr. Lisandro Zambenedetti Granville, agradeço por sempre me apoiarem e me orientarem em tudo que foi possível. Agradeço também aos meus colegas e amigos do projeto e a todos da Rede Nacional de Ensino e Pesquisa (RNP) com quem tive o privilégio de ter contato. Foi incrível ter a oportunidade de trabalhar com vocês e poder aprender novas tecnologias. Grande parte deste trabalho somente foi possível por todo o conhecimento adquirido durante o desenvolvimento do projeto.

A todos os outros grandes professores com quem tive o privilégio de ter contato durante a graduação. Vocês foram essenciais para o meu aprendizado durante essa caminhada. Gostaria de agradecer também a todos funcionários do Instituto de Informática da UFRGS por propiciar um ambiente agradável e acolhedor para os alunos.

Por fim, a todas as outras pessoas que de alguma forma torceram por mim e desejaram o meu sucesso nessa jornada.

RESUMO

FPGAs têm sido cada vez mais utilizadas para o desenvolvimento de projetos envolvendo processamento de sinais e comunicações, computação de alto desempenho, processamento de imagens e visão computacional. Contudo, o custo elevado desses dispositivos pode ser um empecilho para estudantes e pesquisadores. Diante disso, neste trabalho foi desenvolvido um sistema que possibilita o uso remoto de placas de prototipagem equipadas com FPGAs. Esse sistema facilita a programação e coleta de dados, uma vez que toda a infraestrutura necessária já está disponível. O usuário precisa apenas enviar o arquivo sintetizado para a placa. Foram utilizadas ferramentas modernas e sofisticadas para o desenvolvimento, além de uma arquitetura de microsserviços e em camadas para permitir uma boa modularidade.

Palavras-chave: FPGA. Programação remota. Placas de prototipagem. Sistemas digitais.

SPIDER: An infrastructure for remote programming and data collection of FPGAs

ABSTRACT

FPGAs have been increasingly used for the development of projects involving signal processing and communications, high-performance computing, image processing and computer vision. However, the high cost of these devices can be an obstacle for students and researchers. Therefore, in this work a system was developed that allows the remote use of prototyping boards equipped with FPGAs. This system facilitates programming and data collection, since all the necessary infrastructure is already available. The user only needs to send the synthesized file to the board. Modern and sophisticated tools were used for development, in addition to a microservices and layered architecture to allow good modularity.

Keywords: FPGA, remote programming, Prototyping boards, Digital systems.

LISTA DE ABREVIATURAS E SIGLAS

PoMaM - Módulo Power Management Module

FSerDaC- FPGA Serial Data Collector

RPSCon - Remote Power Supply Control

ReTiCap - Real Time Video Capture

DevMan - Device Manager

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O uso de placas de prototipagem equipadas com FPGAs em instituições de ensino e pesquisa na área de sistemas digitais é de extrema importância, pois permite que alunos possam realizar experimentos reais em hardware e dessa forma obter um melhor aprendizado (KARYONO; WICAKSANA, 2013). No caso dos pesquisadores, essa tecnologia tem papel fundamental já que permite a prototipação de sistemas digitais personalizados rapidamente, possibilitando a realização de experimentos com alto desempenho e eficiência energética.

No entanto, estudantes e pesquisadores nem sempre têm acesso aos dispositivos desejados, uma vez que existe uma variedade de modelos com especificações distintas. Muitas vezes, as instituições não possuem esses equipamentos devido ao alto custo que eles possuem.

A solução proposta nesse trabalho é criar uma infraestrutura chamada SPIDER que permite alunos e pesquisadores utilizarem placas com FPGAs remotamente e também permitir que seja possível buscar laboratórios físicos que disponham desses recursos. Instituições parceiras podem conectar suas placas com FPGAs ociosas no sistema e permitir que outras pessoas utilizem esses equipamentos em suas pesquisas. Dessa forma, a SPIDER aumenta a capacidade de colaboração entre as instituições, o acesso à informação e a equipamentos e possibilita o uso remoto de dispositivos em laboratórios permitindo que os usuários possam programar e coletar dados reais de execução gerados pelas placas.

No Capítulo 2, será mostrada a fundamentação teórica por trás da solução proposta. Será discutido sobre alguns trabalhos feitos relacionados a essa área e suas limitações.

Neste trabalho, a arquitetura foi projetada com o objetivo do sistema ser escalável e eficiente. Para isso, são utilizados microsserviços e contêineres em conjunto com tecnologias modernas. A arquitetura e todas as tecnologias utilizadas serão mostradas no Capítulo 3

Inicialmente foi construída uma estrutura visando acomodar com segurança as placas com FPGA, em seguida, foi desenvolvido um sistema completo para o gerenciamento de energia das placas, para evitar que elas necessitassem ficar ligadas o tempo todo.

Após a parte física ser construída, iniciou-se o desenvolvimento dos microsserviços e serviços. Foram desenvolvidos microsserviços responsáveis por gerenciar o fluxo

para a programação da placa, por controlar remotamente o sistema de gerenciamento de energia, por coletar os dados gerados pela placa, por captar imagens em tempo real. Também foram desenvolvidos serviços para permitir o gerenciamento completo dos dispositivos conectados em cada laboratório e para permitir o acesso externo a todo o sistema.

Uma ferramenta chamada Mapa de Inventário foi utilizada como base para o *front-end*. Informações de placas de prototipagem com FPGAs foram modeladas para serem representadas na ferramenta e uma nova funcionalidade foi desenvolvida no Mapa de Inventário para permitir que usuários pudessem ter uma interface intuitiva para programar e coletar resultados dessas placas.

O desenvolvimento completo da solução também pode ser visto com detalhes no Capítulo 3

Com todos os sistemas desenvolvidos eles foram integrados e iniciou-se os testes para validar o funcionamento. O Capítulo 4 mostra com detalhes o funcionamento do sistema e os resultados obtidos.

Por fim, no Capítulo 5 são feitas as considerações finais, descrevendo limitações do sistema e sugestões de trabalhos fututos que podem agregar na evolução do sistema.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 *Field-Programmable Gate Array*

Field-Programmable Gate Arrays (FPGAs) são dispositivos lógicos programáveis que permitem a implementação de circuitos digitais personalizados, oferecendo flexibilidade, desempenho e eficiência energética (KUON; ROSE, 2006). Os FPGAs são compostos por uma matriz de blocos lógicos configuráveis, interconexões programáveis e memórias embutidas, permitindo que sejam implementados diversos tipos de circuitos digitais (HAUCK; DEHON, 2010).

Algumas das aplicações de FPGAs são processamento de sinais, comunicações, criptografia, controle de sistemas e computação de alto desempenho. Por ser um equipamento reconfigurável, os FPGAs oferecem vantagens significativas em relação aos circuitos integrados específicos de aplicação (ASICs), como menor tempo de desenvolvimento, menor custo para pequenas produções e a capacidade de atualizar ou modificar funcionalidades após a fabricação (TRIMBERGER, 2012).

O projeto de circuitos digitais utilizando FPGAs pode ser desafiador, devido à complexidade e à diversidade dos recursos disponíveis (BETZ; ROSE; MARQUARDT, 2012). Ferramentas de projeto assistido por computador (CAD), como compiladores de alto nível, mapeadores lógicos e roteadores, são muito utilizados por projetistas na implementação e na otimização de circuitos em FPGAs (HAUCK; DEHON, 2010).

2.2 Instâncias F1 do Amazon EC2

Com o foco em acelerações de hardware personalizadas e tendo um mercado promissor em aplicações de processamento de imagem e vídeo, *big data* e genômica, a Amazon investiu no uso de FPGAs e lançou uma linha de instâncias específica para essa finalidade.

As chamadas instâncias F1, fornecem ambientes de desenvolvimento, nas quais projetos de FPGA podem ser implantados, registrados e fornecidos como uma *Amazon FPGA Image* (AFI), diretamente do *marketplace* da plataforma.

O objetivo é tornar rápido e simples o acesso a esses recursos tal qual qualquer instância em nuvem, oferecendo escalabilidade e flexibilidade de preço, onde o usuário paga apenas pelo tempo em que foi utilizado. (Amazon, 2023)

2.3 Mapa de inventário

O Mapa de inventário vem sendo desenvolvido desde 2020 em uma parceria entre o Instituto de Informática da Universidade Federal do Rio Grande do Sul (INF-UFRGS) e a Rede Nacional de Ensino e Pesquisa (RNP). O objetivo dele é ser genérico, possibilitando que qualquer tipo de informação seja inserida nele. Nesse trabalho, ele será usado para representar, de forma interativa em um mapa, os dispositivos disponíveis para reserva.

2.3.1 Arquitetura

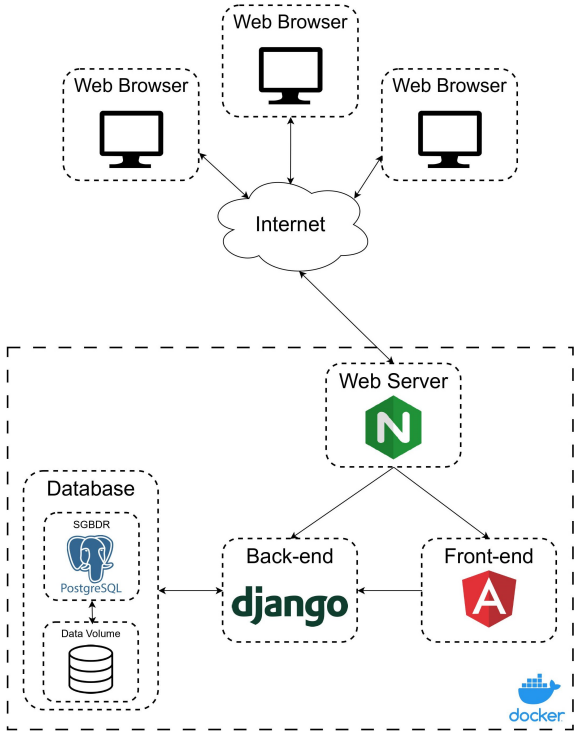
Essa ferramenta foi desenvolvida usando tecnologias robustas, as quais são amplamente utilizadas em desenvolvimento web. Na figura 2.1 é mostrada a arquitetura do sistema. Nela é possível ver que usuários podem acessar a aplicação através de seus navegadores web utilizando a internet. O acesso feito, faz uma requisição ao *web server* (Nginx) e ele redireciona o usuário para a porta de interesse: *back-end* ou *front-end*.

Caso o usuário queira acessar o painel administrativo, o Nginx redireciona para a porta exposta pelo *back-end* (Django). Dessa forma é possível adicionar, editar ou remover informações configurações do mapa. Para isso, o Django se comunica com o banco de dados (PostgreSQL) para realizar as operações necessárias.

Caso o usuário queira acessar o mapa interativo, o Nginx redireciona para a porta exposta pelo *front-end* (Angular). O Angular faz requisições para o Django que por sua vez realiza requisições dos dados brutos para o PostgreSQL. Esses dados são tratados e retornados ao Angular, que exibe essas informações no mapa.

Para manter a rapidez, isolamento, portabilidade e escalabilidade, cada componente da aplicação é executado em um *container* individual Docker. Ou seja, existem quatro *containers* na aplicação, um para o banco de dados, um para o *back-end*, um para o *front-end* e um para o *web server*.

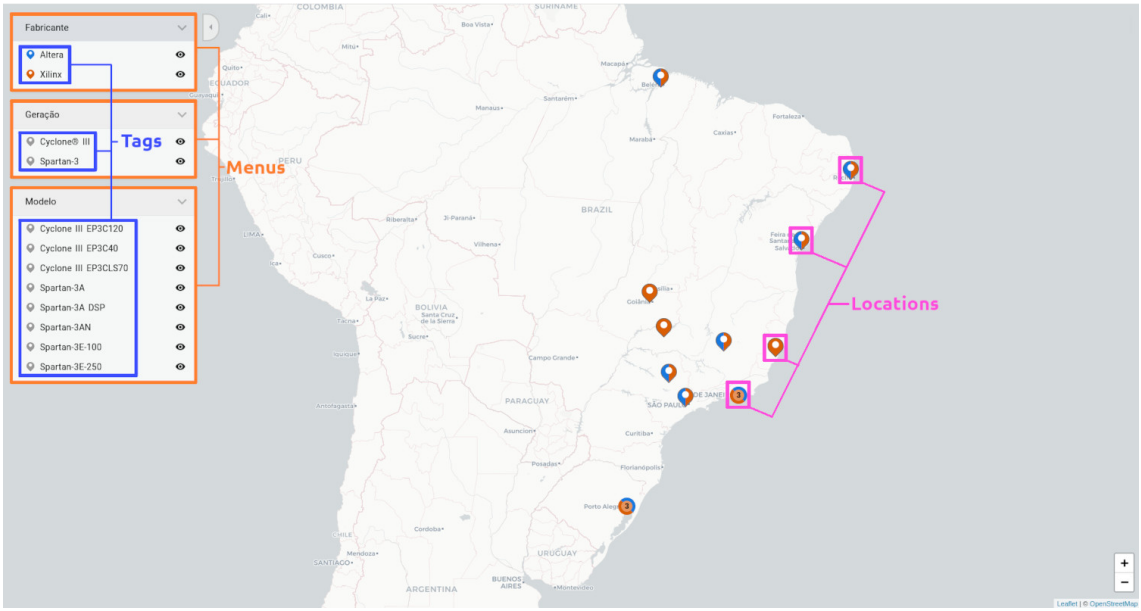
Figura 2.1: Arquitetura do Mapa de Inventário. Fonte: O autor.



2.3.2 Componentes do mapa

A figura 2.2 mostra o mapa do mapa de inventário. Foram inseridos dados de exemplo e feita marcações na imagem para facilitar o entendimento inicial do leitor. Existem três principais componentes: Menus, *Tags* e *Locations*.

Figura 2.2: Mapa de inventário. Fonte: O autor.



Menus podem ser vistos como um conjunto de *tags*. É possível criar diversos menus e definir hierarquias para cada um deles. Neste exemplo, os menus estão marcados em laranja. Eles são: fabricante, geração e modelo.

Tags são marcadores que podem representar qualquer tipo de informação. Elas podem ser atribuídas a um menu, a uma ou mais *locations*, ter uma cor e ter informações que serão exibidas em um *pop-up*. Elas também podem ser filhas de uma ou mais *tags*, dessa forma é possível criar hierarquias para representar qualquer tipo de informação. Essas hierarquias são importantes para a utilização de filtros, que será discutida a seguir. Neste exemplo, as *tags* estão marcadas em azul. No caso das *tags* contidas no menu "fabricante" tem-se: Altera e Xilinx.

Locations são abstrações de localizações. Elas possuem as coordenadas geográficas de uma localização real. Essas coordenadas são utilizadas para posicionar o pino no mapa. Um pino de *location* possui as cores de todas as *tags* contidas nele.

2.3.3 Mudança de contexto

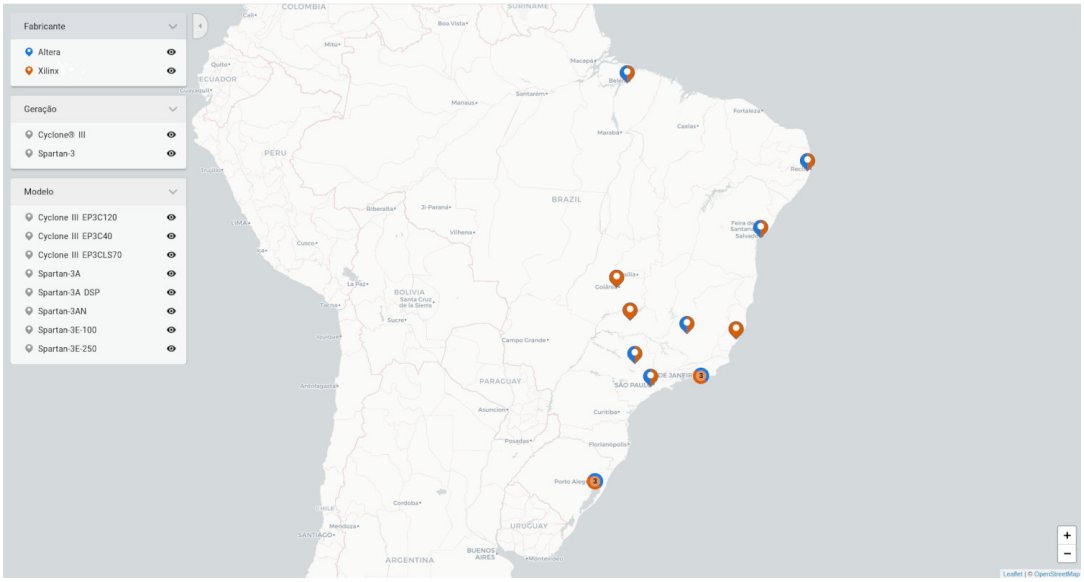
O mapa de inventário possui uma funcionalidade de mudança de contexto. Com ela, é possível que as cores dos pinos do mapa sejam alteradas para o contexto do menu selecionado no momento.

Na figura 2.3a o menu selecionado é "Fabricante". Dessa forma, o mapa exibe os locais onde existem FPGAs dos fabricantes Altera e Xilinx.

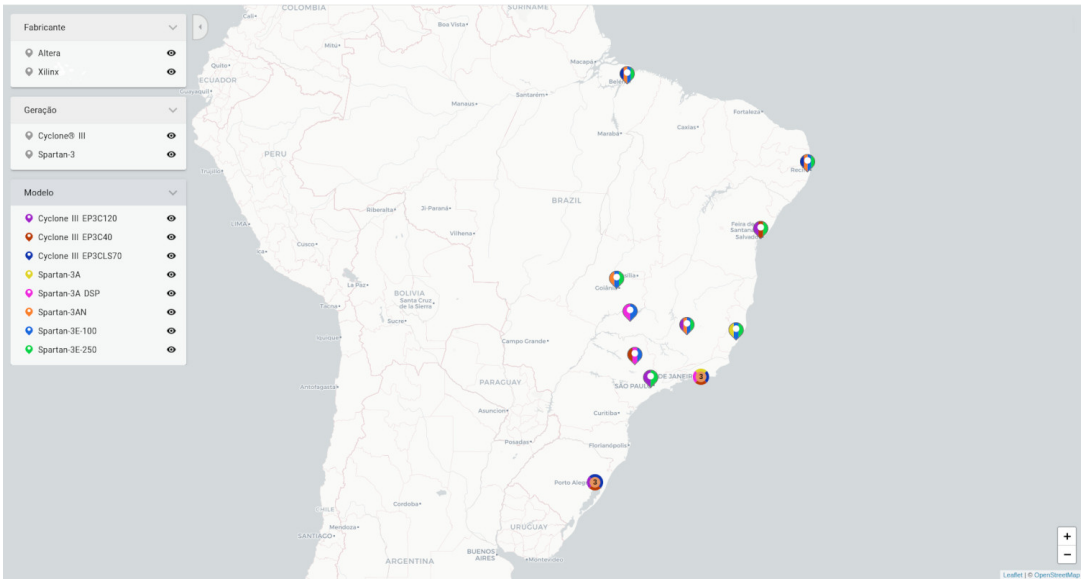
Já na figura 2.3b o menu "Modelo" está selecionado. Assim, o contexto do mapa é modificado e exibe as cores dos pinos conforme os modelos de FPGAs.

Essa funcionalidade é muito importante, pois permite que o usuário busque equipamentos por meio de informações de mais alto nível (como fabricante), ou faça a busca utilizando informações mais específicas (como modelo). Vale lembrar que novos menus poderiam ser criados e adicionado deles informações mais detalhadas, como quantidade de portas lógicas, capacidade de memória, entre outros.

Figura 2.3: Mudança de contexto na visualização das informações. Fonte: O autor.



(a) Contexto de Fabricante.



(b) Contexto de Modelo.

2.3.4 Filtros

O mapa de inventário possui um poderoso recurso de filtros. Com ele é possível filtrar *tags* e propagar esse filtro para todos os filhos da *tag* filtrada.

Figura 2.4: Utilização de filtros no mapa. Fonte: O autor.

(a) Filtrando por equipamento da fabricante Xilinx.

(b) Filtrando por equipamento do modelo Cyclone III EP3CLS70.

Na figura 2.4a é feito um filtro para exibir no mapa somente os locais onde existem equipamentos da fabricante Xilinx. Para esse filtro ser feito, foi clicado no ícone de olho ao lado da *tag* da fabricante Altera. Ao ser feito isso, todas as *tags* que são dependentes de Altera foram ocultadas. É possível observar isso comparando com a figura 2.3a. No menu "Geração" a *tag* "Cyclone III" foi ocultada e no menu "Modelo" as *tags* "Cyclone III EP3C120", "Cyclone III EP3C40" e "Cyclone III EP3CLS70" foram ocultadas.

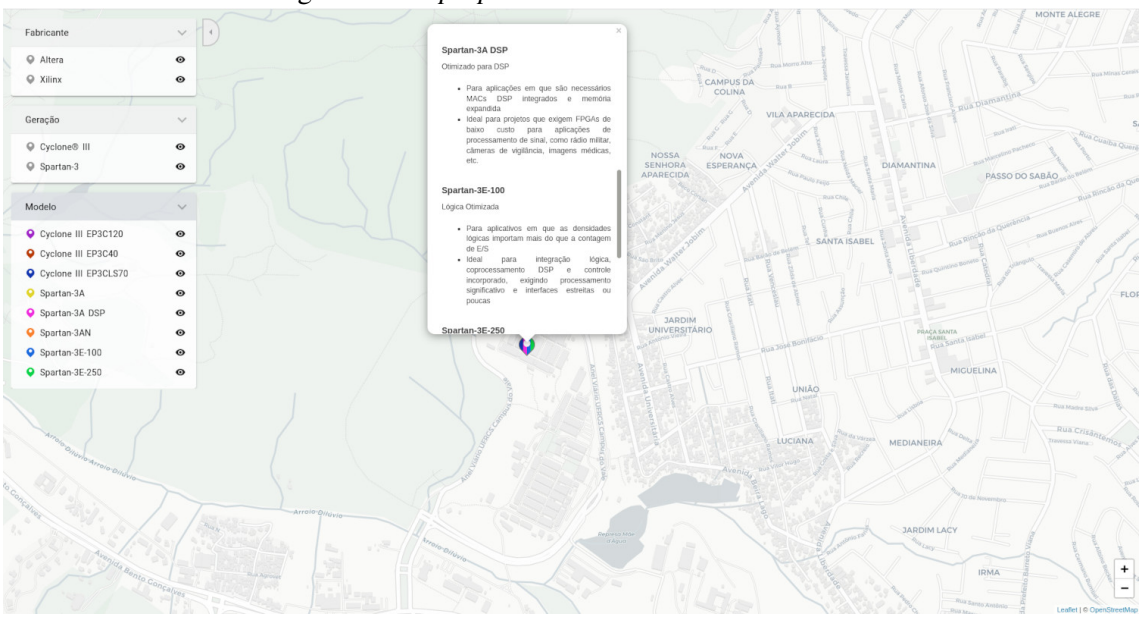
Já na figura 2.4b, é feito um filtro para exibir no mapa somente os locais onde existem equipamentos do modelo "Cyclone III EP3CLS70". Com isso nota-se que todos

os *locations* que não possuem esse modelo foram ocultados, restando no mapa somente os *locations* que contém o equipamento filtrado.

2.3.5 Pop-ups

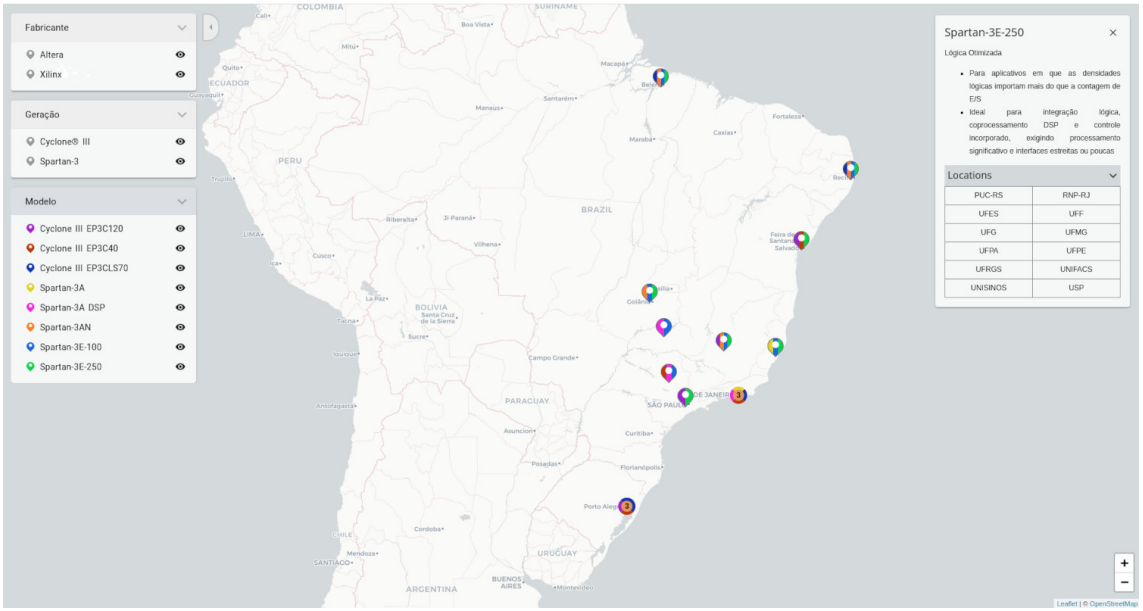
Pop-ups podem ser configurados para cada *location*, permitindo que sejam exibidas informações detalhadas dos equipamentos contidos no *location*. A Figura 2.5 mostra um *pop-up* como exemplo.

Figura 2.5: *Pop-up* de um *location*. Fonte: O autor.



Outro tipo de *pop-up* que pode ser exibido são os de *tags*. A função dele é exibir a descrição detalhada de uma *tag* e mostrar de forma simplificada todas as *locations* que essa *tag* está contida. Para exibi-lo, basta clicar em cima de uma *tag* nos menus. Na Figura 2.6 é mostrado os detalhes da FPGA Spartan-3E-250 e os locais que ela está disponível.

Figura 2.6: *Pop-up* de uma *tag*. Fonte: O autor.



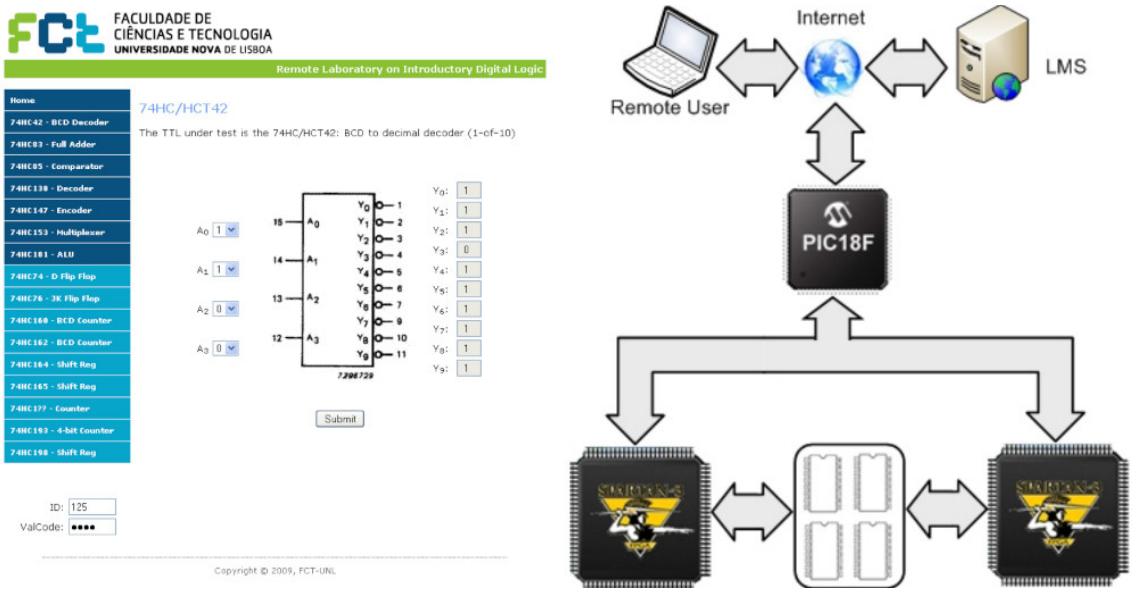
2.4 Trabalhos Relacionados

Na presente seção, os principais trabalhos relacionados a execução remota de código e alocação de recursos armazenados em laboratório de maneira *on-demand* são apresentadas.

Em GOMES et al. (2009), é proposta uma plataforma de laboratório remoto visando ensinar conceitos básicos de circuitos digitais para iniciantes, sem a necessidade de alocar um laboratório físico para isso. Os testes são realizados com o auxílio de duas FPGAs conectadas a um microcontrolador. Uma interface *web* foi criada para habilitar a realização de testes em torno de um conjunto de circuitos combinacionais e sequenciais predeterminados. Os resultados desses testes podem ser visualizados ao lado do símbolo esquemático ou por meio de diagramas de temporização. A principal limitação dessa proposta é que seu acesso é limitado apenas à *intranet* da universidade em que foi testada, não permitindo que alunos fora desse ambiente a usufruam.

Na Figura 2.7 pode-se observar o símbolo eletrônico de um componente, bem como a seleção de entradas (*inputs*) e a arquitetura proposta no trabalho.

Figura 2.7: Interface e arquitetura. (GOMES et al., 2009)



(a) Interface de interação com a FPGA.

(b) Arquitetura do sistema.

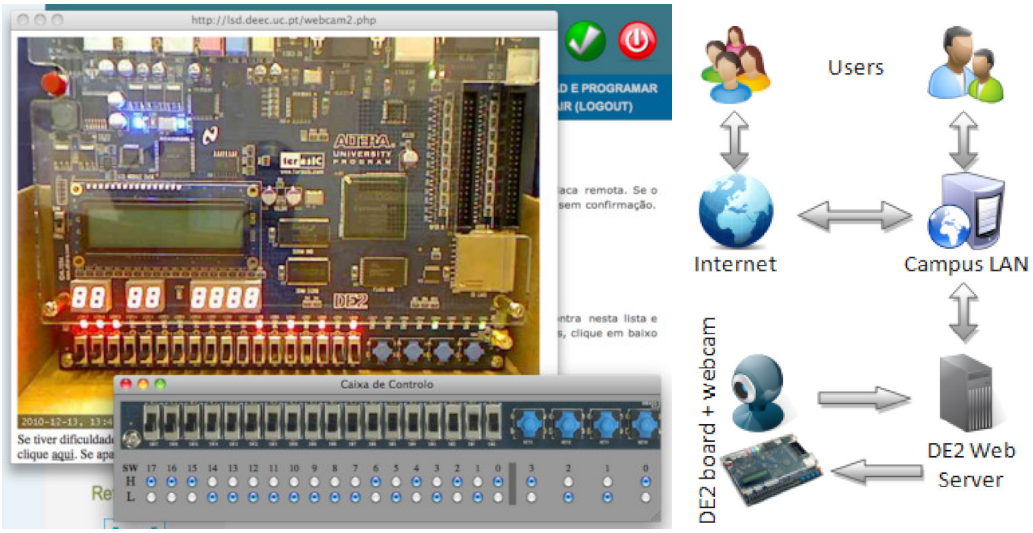
Uma proposta similar a (GOMES et al., 2009) é apresentada em (SOARES; LOBO; DEEC, 2011), também com o propósito de facilitar o acesso de estudantes de cursos introdutórios a testes de circuitos digitais em placas FPGAs, sem a necessidade de deslocamento até um laboratório. Uma interface gráfica simples foi criada para permitir que os usuários pudessem interagir com uma FPGA real a partir de botões virtuais.

O diferencial dessa proposta é que ela oferece além do controle dos componentes por meio de botões, o usuário pode visualizar os testes também através de um *applet* de *webcam*. Uma funcionalidade adicional é a de armazenamento de projetos e testes através de jogos para os estudantes nessa mesma plataforma. A interação do usuário com as funcionalidades apresentadas e a arquitetura proposta é mostrada na Figura 2.8.

Já em (QASSEM et al., 2020), um microsserviço para programar FPGAs remotamente é proposto. O objetivo do trabalho foi proporcionar aos estudantes e pesquisadores acesso fácil e flexível a recursos de FPGA para fins educacionais e de pesquisa. A solução proposta utiliza tecnologias de virtualização (como Docker) e computação em nuvem para permitir o acesso remoto e compartilhado aos dispositivos FPGA, otimizando a utilização dos recursos e reduzindo custos. Nessa proposta, o usuário realiza o *upload* de códigos *Verilog* para o microsserviço, que os implementa na FPGA. A figura 2.9 mostra a arquitetura proposta no trabalho.

Os autores em (QASSEM et al., 2020) também ressaltam a necessidade de se expandir o uso de FPGAs como tecnologias aceleradoras de performance para tarefas cada vez mais voltadas aos campos de inteligência artificial, mais especificamente *Deep*

Figura 2.8: Interface e arquitetura. (SOARES; LOBO; DEEC, 2011)

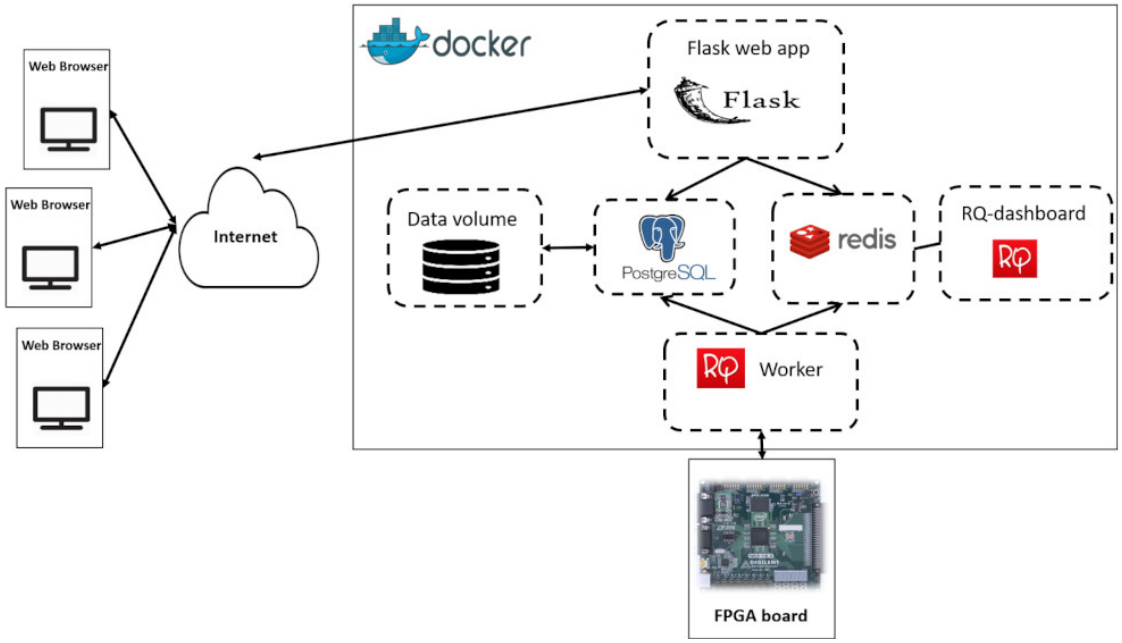


(a) Interface de interação com a FPGA.

(b) Arquitetura do sistema.

Learning, como oportunidades promissoras de exploração futuras.

Figura 2.9: Arquitetura do sistema. (QASSEM et al., 2020)



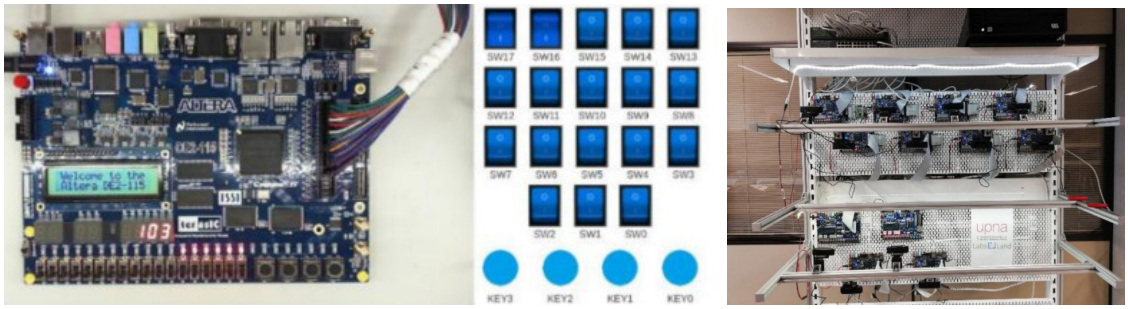
Em (MAYOZ et al., 2020), os autores descrevem a experiência de um laboratório remoto compartilhado entre duas universidades, a Universidade Pública de Navarra (UPNA), na Espanha, e a Universidade Federal de São Paulo (UNIFESP), no Brasil. O laboratório remoto tem como foco o ensino e a pesquisa em sistemas embarcados baseados em FPGA.

Como motivação do trabalho, cita-se a crescente demanda por profissionais qualificados em sistemas embarcados baseados em FPGA e a necessidade de promover a coo-

peração internacional no ensino superior. Também são descritos os recursos de hardware e software, incluindo os servidores utilizados para hospedar o laboratório, as plataformas FPGA, as ferramentas de desenvolvimento e os sistemas de comunicação para conectar os alunos e professores das duas universidades.

Os autores compartilham os resultados positivos alcançados com a implementação do laboratório remoto, incluindo a melhoria na qualidade do ensino e aprendizagem, maior motivação e engajamento dos alunos, e a promoção de colaborações internacionais entre as universidades envolvidas. A figura 2.10 mostra a interface e a arquitetura proposta no trabalho.

Figura 2.10: Interface e infraestrutura de FPGAs. (MAYOZ et al., 2020)

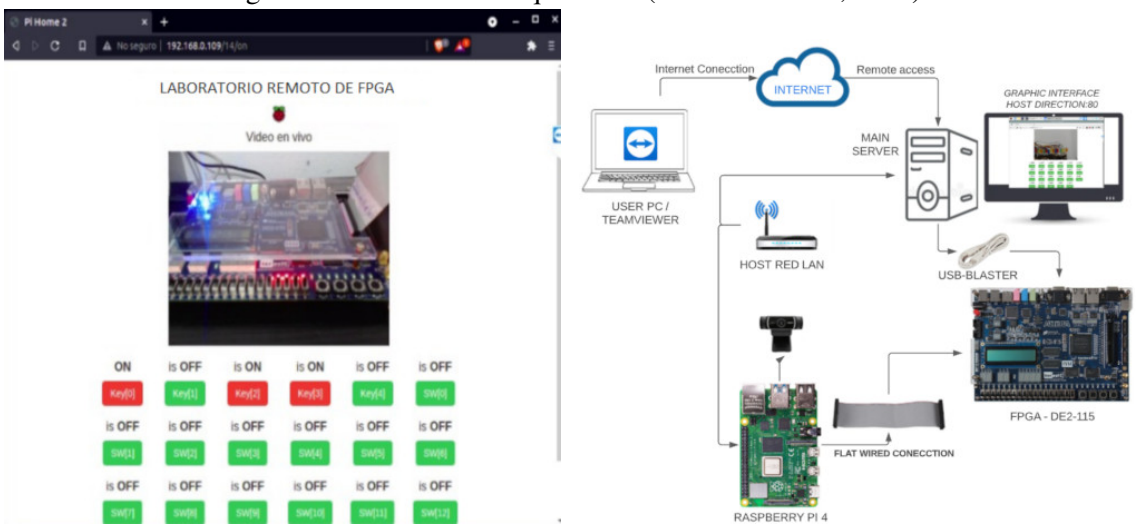


(a) Interface de interação com a FPGA. (b) Infraestrutura de FPGAs.

Em (ABANTO et al., 2022), é apresentado o desenvolvimento e implementação de uma interface web para laboratórios remotos, focada no ensino de engenharia e uso de placas FPGA. A solução utiliza uma placa FPGA-DE2-115 e Raspberry PI 4 para conexão e configuração, podendo ser estendida a outros tipos de placas FPGA. A interface web, desenvolvida com Flask-Python, é intuitiva e interativa, permitindo simular o controle de interruptores e botões físicos e visualizar resultados de experimentos. Essa abordagem proporciona uma contribuição educacional significativa em laboratórios de microeletrônica ou sistemas digitais com controle remoto. O código da interface está disponível no Github para melhorias e uso acadêmico. A figura 2.11 mostra a interface e a arquitetura proposta no trabalho.

Paralelamente, (BLOCHWITZ et al., 2022) apresenta o RemEduLa (*Remote Educational Laboratory*). Um laboratório remoto para ensino de design de hardware utilizando FPGA. O objetivo é proporcionar uma experiência de aprendizado próxima ao ensino presencial, conectando a placa FPGA a um servidor de hardware e permitindo a instrumentação virtual por meio de uma interface web. O sistema oferece controle total sobre a placa de desenvolvimento FPGA e seus periféricos, além de feedback visual em tempo real através de um *stream* de vídeo. O RemEduLa pode ser utilizado em diversas

Figura 2.11: Interface e arquitetura. (ABANTO et al., 2022)

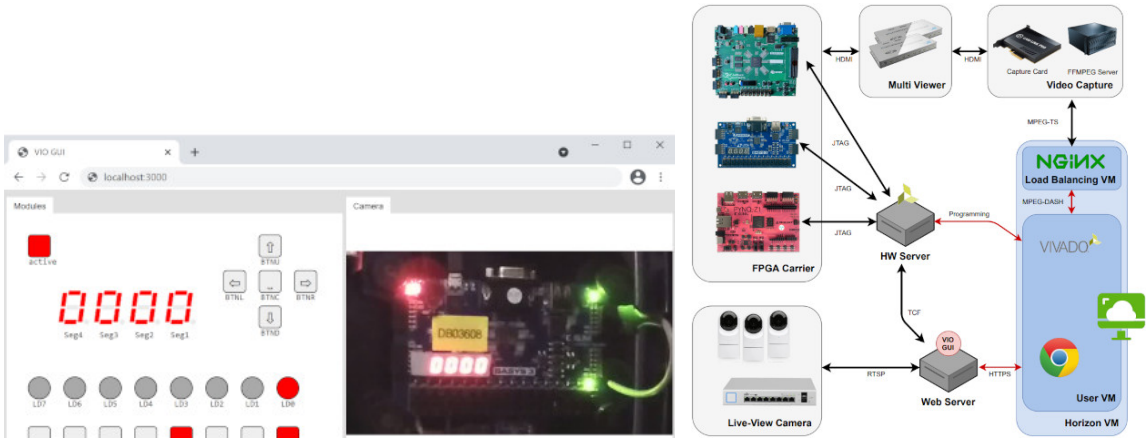


(a) Interface de interação com a FPGA. (b) Arquitetura do sistema.

disciplinas, como linguagem de descrição de hardware (HDL), arquitetura de computadores e sistemas embarcados, e permite acesso às placas FPGA independente dos horários de laboratório presencial na universidade.

A figura 2.12 mostra a interface e a arquitetura proposta no trabalho.

Figura 2.12: Interface e arquitetura. (BLOCHWITZ et al., 2022)



(a) Interface de interação com a FPGA. (b) Arquitetura do sistema.

Um problema desses experimentos citados acima é que eles não retornam um *feedback* de valores da execução. Eles apenas apresentam uma plataforma simplificada de interação com a placa com FPGA, por meio de botões e LEDs. Para alunos que estão iniciando seus estudos, essas ferramentas podem ser suficiente para o aprendizado. Entretanto, para pesquisadores é necessário que mais informações relacionadas a execução sejam retornadas, como tempo de execução, blocos lógicos utilizados, valores resultantes, para que eles possam realizar uma pesquisa mais profunda e detalhada.

3 METODOLOGIA

Nas seções a seguir, será detalhada a metodologia utilizada para o desenvolvimento deste trabalho.

3.1 Tecnologias

Nas subseções a seguir serão mostradas as tecnologias utilizadas para o desenvolvimento do sistema SPIDER.

3.1.1 Django

Para construir o *back-end* do Serviço *SPIDER-Device Manager*, explicado na subseção 3.3.9, foi utilizado o *framework* Django.

Django é um *framework* de desenvolvimento web *back-end* escrito em Python, de alto nível e de código aberto. Ele permite a criação de aplicações web de forma rápida e eficiente, seguindo o padrão de arquitetura Model-View-Template (MVT) (Django Software Foundation, 2023a). A sua filosofia se baseia no conceito de *"Don't Repeat Yourself"* (DRY), incentivando a reutilização de código e a modularidade (HOLOVATY; KAPLAN-MOSS, 2009). Isso resulta em um desenvolvimento mais eficiente, menos propenso a erros e mais fácil de manter (VINCENT, 2021).

O Django oferece uma série de componentes integrados que abordam questões comuns do desenvolvimento web, como autenticação de usuário, gerenciamento de sessão, mecanismos de banco de dados, manipulação de formulários, e suporte a internacionalização (i18n) (RAVINDRAN, 2015). Esses componentes são desenvolvidos com foco na segurança, ajudando a evitar problemas como *Cross Site Scripting* (XSS), *Cross Site Request Forgery* (CSRF) e Injeção de SQL (Django Software Foundation, 2023b).

A comunidade ativa e crescente do Django é outra característica importante, que contribui com inúmeros pacotes adicionais e suporte contínuo para os desenvolvedores. O Django também conta com uma documentação abrangente, que facilita o aprendizado e a adoção do *framework* (RAVINDRAN, 2015).

Em termos de desempenho, o Django se mostra capaz de lidar com cargas de trabalho variadas e escalonar conforme a demanda, o que o torna adequado para uma

ampla gama de aplicações, desde pequenos projetos até sistemas empresariais complexos (VINCENT, 2021).

3.1.2 Django REST *framework*

O Django REST *Framework* (DRF) é utilizado neste trabalho para fazer a criação de uma API simples e eficaz para o Serviço *SPIDER-Device Manager*, explicado na subseção 3.3.9.

Este *framework* é uma extensão poderosa do Django, que facilita a construção de APIs web de maneira rápida e flexível. O DRF fornece ferramentas para serializar dados, lidar com autenticação e permissões, e oferece vistas e *class-based views* que tornam a criação de *endpoints* de uma API muito mais simples e intuitiva. Com o DRF, os desenvolvedores podem criar *APIs RESTful* de alto desempenho com menos esforço, aproveitando ao máximo as funcionalidades do Django (Tom Christie, 2023).

3.1.3 Angular

Angular é um popular *framework* de desenvolvimento web *front-end* de código aberto, mantido pelo Google, utilizado para criar aplicações web de página única (*Single Page Applications*) (Angular Team, 2023). O Angular é construído com base no TypeScript, uma linguagem de programação tipada e de alto nível, que é um superconjunto do JavaScript. A arquitetura do Angular é baseada em componentes, favorecendo a reutilização de código, a modularidade e a escalabilidade (ULUCA, 2018).

O Angular apresenta diversas funcionalidades, como suporte para a construção de interfaces de usuário responsivas, manipulação de eventos, injeção de dependências e comunicação com *APIs RESTful*. Ele também possui uma série de ferramentas e bibliotecas integradas, como o Angular Material, que oferece uma coleção de componentes de interface do usuário seguindo as diretrizes do Material Design (Google, 2023).

Em termos de desempenho, o Angular utiliza técnicas como a detecção de mudanças e a otimização de compilação para melhorar a velocidade e eficiência das aplicações. Além disso, a comunidade Angular é vasta e ativa, proporcionando suporte contínuo, pacotes adicionais e atualizações frequentes para o *framework* (ULUCA, 2018).

3.1.4 Nginx

O Nginx é um servidor web de alto desempenho. Ele foi criado por Igor Sysoev em 2002. Foi desenvolvido para resolver o problema da alta concorrência e escalabilidade na internet. O Nginx é conhecido por sua estabilidade, baixo consumo de recursos e capacidade de lidar com milhares de conexões simultâneas, o que o torna ideal para sites com alto tráfego.

O Nginx opera com base em um modelo assíncrono, orientado a eventos, que permite gerenciar eficientemente múltiplas conexões simultâneas sem a necessidade de criar um processo ou *thread* para cada solicitação. Isso faz com que o servidor seja mais rápido e eficiente em comparação a outros servidores web tradicionais, como o Apache.

Além disso, o Nginx pode ser configurado como um proxy reverso, o que permite que ele encaminhe as solicitações dos clientes para outros servidores *back-end* e, em seguida, retorne as respostas para os clientes. Essa função também pode ser usada para balanceamento de carga entre vários servidores, garantindo uma distribuição equilibrada das solicitações e melhorando a disponibilidade e a escalabilidade dos sistemas (NEDELCU, 2013).

3.1.5 PostgreSQL

PostgreSQL é um sistema gerenciador de banco de dados relacional (SGBDR) de código aberto, desenvolvido inicialmente na Universidade da Califórnia em Berkeley no início dos anos 1990. Ele é caracterizado por ser robusto, escalável e altamente extensível, o que o torna adequado para uma ampla variedade de aplicações, desde pequenos projetos até sistemas empresariais.

PostgreSQL é compatível com o padrão SQL (*Structured Query Language*) e oferece muitas funcionalidades avançadas, como suporte para consultas complexas, transações, armazenamento de dados JSON, indexação avançada, *full-text search* e armazenamento de dados espaciais, entre outras.

Outra característica marcante do PostgreSQL é a sua extensibilidade, permitindo que os desenvolvedores criem tipos de dados personalizados, operadores e funções. Além disso, é possível estender a funcionalidade do sistema através da instalação de extensões, que são módulos adicionais que oferecem recursos específicos (MOMJIAN, 2001) (PostgreSQL Global Development Group, 2023).

3.1.6 Docker

Docker é uma plataforma de código aberto criada em 2013 que facilita a automação, o empacotamento e a distribuição de aplicações em contêineres. Ele tem como objetivo simplificar e acelerar o desenvolvimento, os testes e a implantação de aplicações, garantindo que funcionem de maneira uniforme em diferentes ambientes.

Os contêineres são unidades isoladas que acondicionam o software, as bibliotecas e as dependências necessárias para executar uma aplicação. Eles proporcionam um ambiente consistente e controlado, permitindo que as aplicações operem de forma confiável em diversos sistemas e infraestruturas. Contêineres são mais ágeis e leves que as máquinas virtuais tradicionais, uma vez que compartilham o mesmo kernel do sistema operacional e não necessitam de um sistema operacional completo para cada aplicação.

Para gerenciar contêineres e imagens, o Docker utiliza um sistema de gerenciamento de imagens, que são os modelos usados para criar contêineres. Essas imagens são armazenadas em um repositório centralizado chamado Docker Hub, onde desenvolvedores podem compartilhar e obter imagens para suas aplicações.

Além disso, o Docker oferece ferramentas e serviços adicionais que facilitam a implantação e o gerenciamento de aplicações em larga escala, como o Docker Compose, Docker Swarm e Kubernetes. Essas ferramentas ajudam a aprimorar ainda mais a experiência dos desenvolvedores ao trabalhar com contêineres e suas aplicações (NICKOLOFF; KUENZLI, 2019).

3.1.7 Docker Compose

Para a definição de toda a infraestrutura de contêineres do projeto, foi utilizado o Docker compose.

Docker Compose é uma ferramenta utilizada para definir e executar aplicações Docker *multi-container*. Ele permite aos usuários definir, em um único arquivo, todas as configurações de serviços, redes e volumes de que uma aplicação precisa, facilitando tanto o processo de desenvolvimento quanto o de implantação de aplicações complexas que dependem de vários serviços para funcionar corretamente.

A principal vantagem do Docker Compose é a capacidade de codificar a infraestrutura e as dependências de uma aplicação em um formato declarativo. O arquivo central utilizado pelo Docker Compose é chamado *docker-compose.yml*. Nesse arquivo, os de-

senvolvedores especificam os serviços que compõem a aplicação, os volumes e as redes que serão utilizados, bem como outras configurações relevantes.

Ao utilizar o Docker Compose, a gestão da aplicação torna-se muito mais simples. Por exemplo, com um único comando, *docker-compose up*, é possível iniciar todos os serviços definidos no arquivo de configuração em sua ordem correta. Isso garante que, independentemente da complexidade da aplicação, ela possa ser replicada, testada e implantada de maneira consistente em diferentes ambientes, desde a máquina de um desenvolvedor até servidores de produção.

Além disso, o Docker Compose também auxilia no escalonamento dos serviços e na interligação entre contêineres, garantindo que eles possam se comunicar entre si de maneira eficiente. Para equipes de desenvolvimento, isso se traduz em uma maneira mais simplificada e robusta de gerenciar e orquestrar contêineres, tornando todo o processo de desenvolvimento, teste e implantação muito mais ágil e confiável (Docker Inc., 2023).

3.1.8 *Python*

O projeto foi desenvolvido utilizando a linguagem de programação *Python*, uma linguagem de alto nível, interpretada, e de propósito geral, que foi criada por Guido van Rossum e lançada pela primeira vez em 1991. Desde sua criação, o *Python* cresceu em popularidade devido à sua legibilidade, simplicidade e versatilidade, tornando-se uma das linguagens de programação mais utilizadas no mundo. (FOUNDATION, 2023)

A filosofia central do *Python* é destacada pelo "Zen do *Python*", um conjunto de aforismos que captura a essência do design da linguagem. Uma das suas máximas mais famosas é "A legibilidade conta", o que reflete a ênfase da linguagem em código claro e fácil de entender (PETERS, 2023). O *Python* favorece uma sintaxe tanto limpa quanto expressiva, permitindo que os programadores expressem conceitos complexos com menos linhas de código em comparação com muitas outras linguagens.

Outra força notável do *Python* é sua natureza extensível e a vasta biblioteca padrão que oferece. Isso permite que os programadores realizem uma ampla gama de tarefas, desde automação de processos simples até desenvolvimento de aplicações web complexas, passando por análise de dados e aprendizado de máquina. Seu ecossistema rico é composto por milhares de pacotes e *frameworks* de terceiros disponíveis através do *Python Package Index (PyPI)*, o que facilita ainda mais a expansão das capacidades padrão da linguagem (AUTHORITY, 2023).

Na era moderna, o *Python* encontrou aplicação em diversas áreas: desenvolvimento web com *frameworks* como *Django* e *Flask*; ciência de dados e aprendizado de máquina com bibliotecas como pandas, *numpy* e *TensorFlow*; automação com *scripts* simples; desenvolvimento de aplicações de *desktop* e até mesmo em sistemas embarcados e *IoT* (SAABITH; VINOTHRAJ; FAREEZ, 2020).

3.1.9 *Flask*

O *Flask* é um *microframework* de desenvolvimento *web* projetado em *Python*. Denominado "micro"não devido a limitações em sua capacidade, mas porque mantém um núcleo simples e extensível, ele oferece aos desenvolvedores a liberdade de escolher como desejam implementar certas funcionalidades em suas aplicações. A simplicidade e a adaptabilidade do *Flask* o tornaram uma escolha popular para desenvolvedores *web* que desejam criar aplicações *web* rápidas e eficientes (PALLETS, 2023a).

O *Flask* oferece o essencial para a criação de aplicações *web*, não impondo uma estrutura específica de projeto. Isso confere aos desenvolvedores a liberdade de escolher as ferramentas e bibliotecas que melhor atendem às suas necessidades. Possui várias extensões disponíveis que facilitam a adição de funcionalidades, como manipulação de formulários, autenticação de usuários e integração com bancos de dados. Adicionalmente, o *Flask* é baseado no padrão *WSGI* (*Web Server Gateway Interface*) do Python, uma interface que promove a interoperabilidade entre servidores *web* e aplicações ou *frameworks*. (DUMPLETON; EBY, 2023).

O *Flask* utiliza rotas para mapear *URLs* para funções de visualização. Essas funções são responsáveis por processar os dados e retornar uma resposta ao cliente (PALLETS, 2023c). Em termos de apresentação, o *Flask* utiliza o mecanismo de *template Jinja2*, permitindo aos desenvolvedores criar páginas *web* dinâmicas com uma sintaxe clara e concisa. Uma vantagem adicional para os desenvolvedores é o servidor de desenvolvimento integrado ao *Flask*, facilitando o teste e o desenvolvimento de aplicações (PALLETS, 2023b).

Desta forma, o *Flask* é uma ferramenta poderosa e versátil que desempenhou um papel crucial na popularização do desenvolvimento *web* em *Python*. Sua simplicidade, aliada à sua capacidade de ser estendido conforme necessário, o torna uma excelente escolha para projetos que vão desde pequenas aplicações até sistemas *web* complexos.

3.1.10 Arduino

Arduino é uma plataforma de prototipagem eletrônica de código aberto projetada para tornar a eletrônica mais acessível para entusiastas e qualquer pessoa interessada em criar projetos interativos.

A plataforma combina tanto hardware, na forma de placas de circuito impresso físicas, quanto software, com uma IDE (Ambiente de Desenvolvimento Integrado) que permite a programação dessas placas. O ambiente de programação, baseado em C/C++, é projetado para ser intuitivo e fácil de usar, tornando-o acessível mesmo para aqueles que não têm experiência prévia em programação ou eletrônica (UNICAMP, 2023).

Uma das principais forças do Arduino é sua versatilidade. As placas são equipadas com entradas e saídas digitais e analógicas, permitindo a interação com uma variedade de componentes, desde LEDs simples e botões até sensores sofisticados e módulos de comunicação. Isso dá aos criadores a capacidade de desenvolver projetos que podem sentir e reagir ao ambiente, estabelecendo a base para uma vasta gama de aplicações, desde robótica e automação doméstica até arte interativa e instalações sonoras.

Em resumo, o Arduino é mais do que uma simples placa ou plataforma; é uma iniciativa que transformou a face da educação em eletrônica, prototipagem e fabricação digital. Continua a inspirar e habilitar pessoas de todas as idades e origens a se tornarem criadores, inovadores e solucionadores de problemas (ARDUINO, 2023).

3.1.11 *Conversor Serial-USB* CP2102

Neste projeto foi utilizado um módulo CP2102, conversor serial-USB produzido pela *Silicon Labs*. Esta peça de *hardware* permite facilitar a comunicação entre dispositivos que utilizam comunicação serial e computadores que, em sua maioria, são equipados predominantemente com portas USB.

O núcleo da funcionalidade do CP2102 está em sua habilidade de transformar os dados da comunicação serial (geralmente *UART*) em dados USB e vice-versa. Isso é crucial em muitas aplicações, desde a programação de microcontroladores até a comunicação com módulos de GPS, sensores e outros dispositivos periféricos.

A versatilidade do CP2102 se manifesta em suas características adicionais, como pinos de controle de fluxo e capacidade de configuração de parâmetros como taxa de transmissão e bits de parada. Isso oferece aos usuários uma flexibilidade considerável

para se adaptar a diferentes requisitos de comunicação (LABS, 2023).

No entanto, como qualquer componente eletrônico, o CP2102 não está livre de desafios. A seleção correta de parâmetros de comunicação e a compatibilidade com dispositivos específicos podem, por vezes, exigir uma compreensão mais aprofundada e uma configuração minuciosa.

3.1.12 Microsserviços

Nesse projeto, foi utilizado uma arquitetura de microsserviços, visando garantir uma boa modularidade, facilitar a manutenibilidade e garantir uma escalabilidade eficaz.

A arquitetura de microsserviços tem emergido como uma abordagem inovadora na concepção de sistemas distribuídos, oferecendo uma solução para desenvolvimento e manutenção de aplicações escaláveis, resilientes e facilmente gerenciáveis. Este conceito é fundamentado na decomposição de uma aplicação em pequenas partes autônomas que se comunicam por meio de interfaces bem definidas, tipicamente usando protocolos leves, como HTTP/REST.

O estudo *'Microservices: Yesterday, Today, and Tomorrow'* (DRAGONI et al., 2017), destaca que a principal motivação por trás da adoção dos microsserviços é a necessidade de lidar com a crescente complexidade dos sistemas e a rapidez exigida nas entregas de software. Em vez de uma única e monolítica base de código, os microsserviços permitem que equipes diferentes desenvolvam, testem e implantem serviços de forma independente, facilitando a manutenção e a escalabilidade.

Uma das principais vantagens da arquitetura de microsserviços é a possibilidade de escalar serviços individuais de acordo com a demanda, sem a necessidade de escalar todo o sistema (NEWMAN, 2021).

3.1.13 Relé

Neste trabalho foi utilizado um módulo com 8 relés para possibilitar que fontes externas possam energizar ou desenergizar as placas com FPGA.

O relé é um dispositivo eletromecânico que opera como um interruptor controlado por um circuito elétrico. Sua principal função é permitir que um circuito de baixa potência controle um circuito de potência superior. No núcleo de um relé, encontra-se uma bobina

de fio enrolada ao redor de um núcleo de ferro. Ao passar corrente elétrica por esta bobina, um campo magnético é gerado. Este campo magnético, por sua vez, atrai uma alavanca metálica responsável por mudar o estado dos contatos elétricos do relé. (Basic Electronics Tutorials, 2022)

Os contatos de um relé podem ser classificados como normalmente abertos (NA) ou normalmente fechados (NF). Se, por exemplo, um relé tem contatos que são normalmente abertos, estes contatos fecharão e conectarão o circuito quando a bobina for energizada. Se os contatos forem normalmente fechados, ocorrerá o oposto; os contatos se abrirão e desconectarão o circuito quando a bobina for energizada. (Basic Electronics Tutorials, 2022)

3.2 Arquitetura do sistema

A arquitetura do sistema foi estruturada em camadas visando maior modularidade, o que facilita a evolução e manutenção do projeto. As seis camadas definidas são:

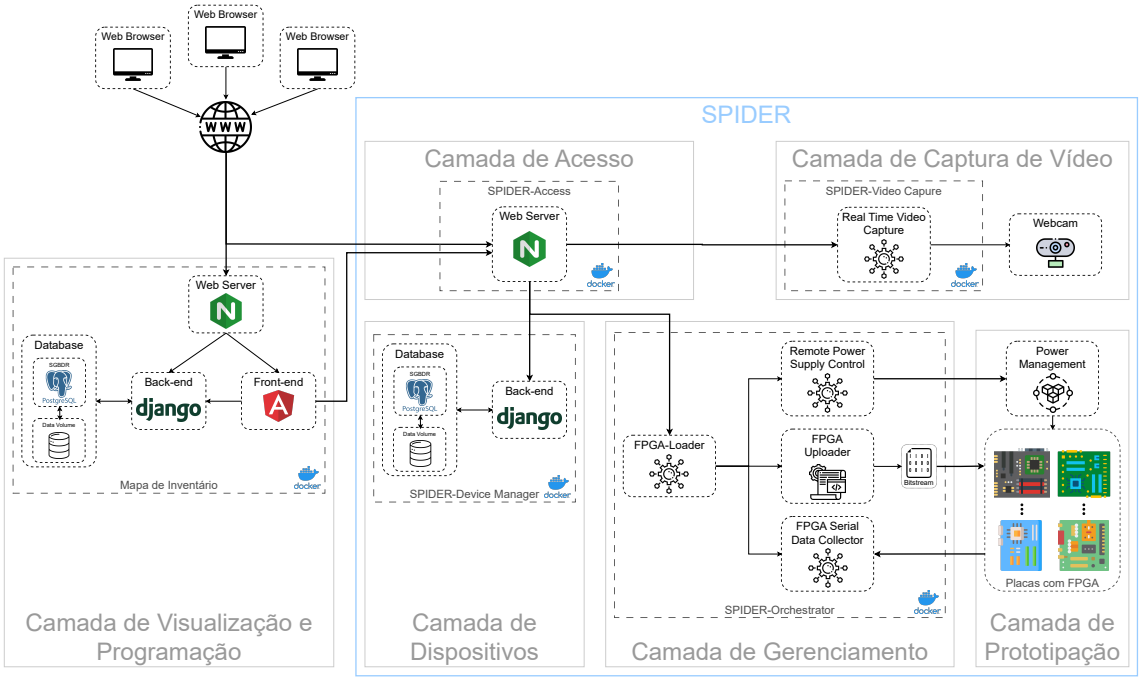
1. Camada de Visualização e Programação: Fornece uma visualização detalhada, intuitiva e dinâmica de dispositivos disponíveis para programação remota ou para uso presencial. Filtros podem ser aplicados para facilitar a busca por dispositivos específicos. Nesta camada, é possível selecionar uma placa, programá-la remotamente e coletar os resultados de execução.
2. Camada de Acesso: Responsável por conectar o mundo externo com o sistema SPIDER. Ela permite que a Camada de Visualização e Programação acesse a infraestrutura e utilize todos os recursos disponíveis. Além disso, oferece aos administradores de laboratórios remotos a capacidade de gerenciar dispositivos físicos, tornando-os disponíveis para uso à distância.
3. Camada de Dispositivos: Fornece dados atualizados e relevantes dos dispositivos disponíveis para serem utilizados remotamente. Além disso, permite que a Camada de Gerenciamento consuma esses dados para gerenciar o fornecimento de energia dos dispositivos.
4. Camada de Gerenciamento: Realiza o gerenciamento de todos os dispositivos conectados ao sistema. Ela possibilita o controle do fornecimento de energia, a programação e coleta de dados dos dispositivos individualmente.
5. Camada de Prototipação: Possui os dispositivos físicos do sistema. Nela, placas

de prototipação equipadas com FPGA podem ser integradas ao sistema e receber alimentação através de portas específicas para o fornecimento de energia.

6. Camada de Captura de Vídeo: Responsável por fornecer imagens em tempo real dos dispositivos físicos.

Para a camada 1, utiliza-se Mapa de Inventário (2.3) como base. Foi implementado uma nova funcionalidade para criar uma página dedicada à programação de placas, proporcionando uma experiência de usuário simplificada e intuitiva. Por outro lado, as camadas 2, 3, 4, 5 e 6 fazem parte da infraestrutura construída inteiramente para o serviço SPIDER. A Figura 3.1[1] apresenta de forma completa e detalhada a arquitetura do sistema, evidenciando cada uma das camadas e serviços envolvidos.

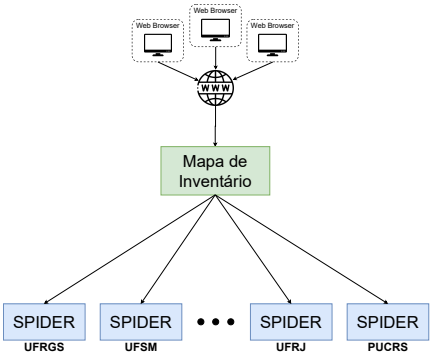
Figura 3.1: Arquitetura do sistema. Fonte: O autor.



A arquitetura adotada possibilita a criação de novas instâncias do sistema SPIDER em diferentes universidades. Como a arquitetura foi projetada utilizando contêineres na sua totalidade, a implantação de uma nova instância em um novo local é extremamente simples. A Figura 3.2[1] mostra um exemplo em que existem diversas universidades executando o SPIDER. Neste trabalho, será utilizado somente uma instância no INF-UFRGS.

[1]Fonte dos ícones da Figura 3.1: Flaticon.com - Autores dos ícones: Ícone 'Script': Eucalyp; Ícones 'Microsserviço', 'Internet', 'Código', 'Hardware' e 'Webcam redonda': Freepik; Ícone 'Módulo': pretty-cons; Ícone 'Big data': juicy_fish; Ícone 'Motherboard': Smashicons; Ícone 'Circuitos': Maan Icons; Ícone 'Computador': Talha Dogar.

Figura 3.2: Arquitetura do sistema. Fonte: O autor.



3.3 Implementação

3.3.1 Módulo *Power Management Module* (PoMaM)

Com uma infraestrutura que permite a programação remota de diversas placas de prototipagem com FPGA, surge um problema. Estas placas precisariam permanecer constantemente energizadas, aguardando a eventualidade de alguém querer programá-las. Isto pode acelerar o desgaste dos componentes eletrônicos das placas, representando um custo significativo para as instituições envolvidas, especialmente considerando o alto valor de muitos desses equipamentos.

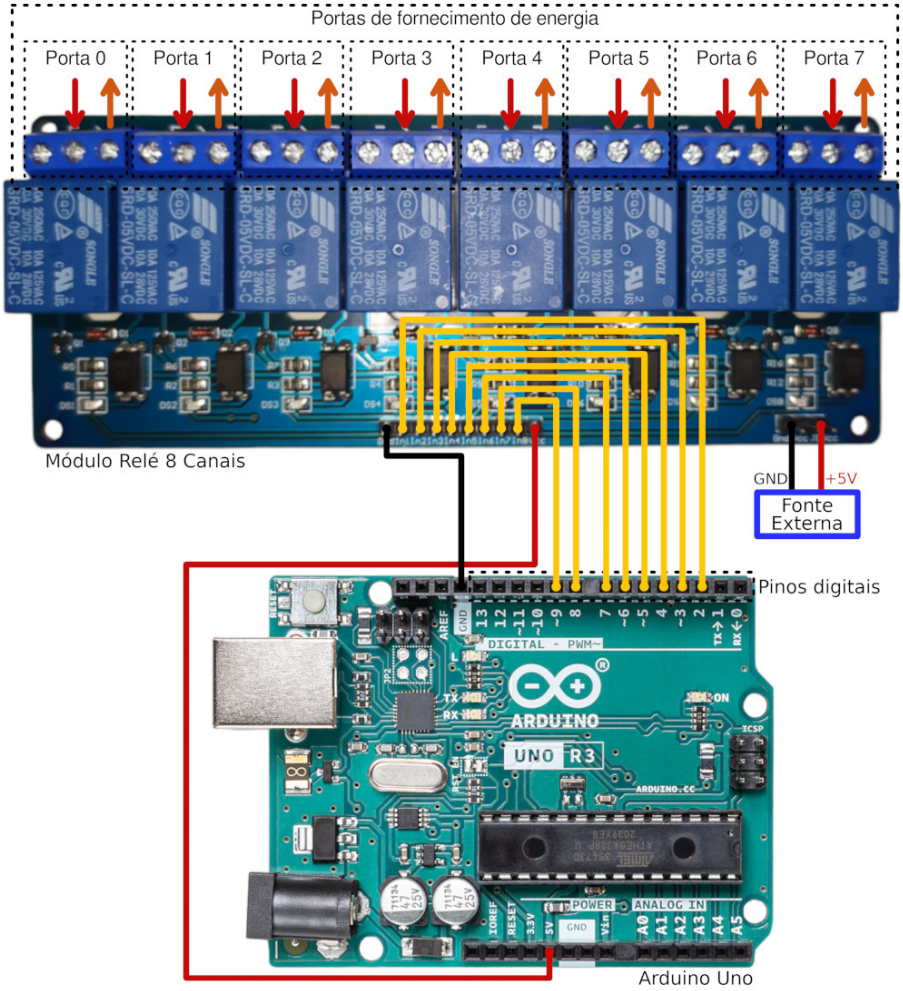
Para solucionar esse problema, foi desenvolvido um módulo de gerenciamento de energia. Essa solução permite energizar ou desenergizar portas de fornecimento de energia individualmente. Ao se comunicar com o Serviço RPSCon 3.3.7, que será explicado com detalhes nas seções a seguir, torna-se possível fazer esse controle de energia remotamente.

Fontes externas podem ser conectadas a cada porta de fornecimento de energia, garantindo a corrente e tensão adequadas para a placa com FPGA conectada. Esta solução foi adotada devido à variedade de modelos de placas, cada um com suas próprias necessidades de alimentação.

Para gerenciar a energia de cada porta individualmente, foi utilizado um Arduino Uno em conjunto com um Módulo Relé de 8 Canais. Os pinos digitais do Arduino (no código *powerSupplyPins*) são conectados aos respectivos pinos de entrada do módulo relé. Como o Arduino não é capaz de fornecer corrente suficiente para acionar todos os relés simultaneamente (caso em que todas as portas são energizadas), o módulo relé é alimentado por uma fonte externa capaz suprir a demanda de energia. A fonte externa

utilizada possui 5 volts de tensão e 1 ampere de corrente. O esquema elétrico pode ser visto na Figura 3.3.

Figura 3.3: Esquema de ligação entre Arduino e Módulo Relé. Imagem do Arduino obtida em: (Arduino, 2023). Fonte: O autor.



Quando é enviado um sinal *LOW* (0V) para um pino de entrada do módulo relé, o relé respectivo é energizado, permitindo que a energia flua do conector de entrada (seta vermelha na figura 3.3) para o conector de saída do relé (seta laranja na figura 3.3)

De forma contrária, quando é enviado um sinal *HIGH* (5V) para um pino de entrada do módulo relé, o relé respectivo é desenergizado, impedindo que a energia flua do conector de entrada (seta vermelha na figura 3.3) para o conector de saída (seta laranja na figura 3.3)

Um programa foi desenvolvido para permitir o gerenciamento de energia e a comunicação com serviços externos. A lógica completa executada no Arduino é apresentada nos algoritmos a seguir.

No momento em que o Arduino é ligado, o Algoritmo 1 é executado. Ele é res-

ponsável por inicializar o equipamento, configurando as portas digitais e iniciando a comunicação serial.

Algoritmo 1 Função de configuração e inicialização do Arduino

```
1: powerSupplyPins ← Define os pinos digiais do Arduino
2: numPowerSuppliers ← Obtém a quantidade de fornecedores de energia
3:
4: Função SETUP
5:     Inicie a comunicação serial com baudRate de 9600
6:     Configure todos os pinos de powerSupplyPins como OUTPUT
7:     Inicie todos os pinos de powerSupplyPins com o valor HIGH    ▷ Envia sinal para desenergizar todas as portas
```

Após ser inicializado, o Algoritmo 2 é executado em *loop*. Ele é responsável por receber comandos via serial, realizar o *parse* e chamar a função correspondente para executar o comando desejado na porta desejada.

Algoritmo 2 Função principal do Arduino

```
1: Função LOOP
2:     enquanto True faça
3:         se Existe dado disponível na conexão serial então
4:             command ← Leia o comando
5:             se command inicia com "on_" então
6:                 se command = "on_all" então
7:                     TURNONALLPOWERSUPPLIERSPORTS()
8:                 senão
9:                     powerSupplyPort ← Extraia o número da porta após "on_"
10:                    TURNONPOWERSUPPLYPORT(powerSupplyPort)
11:            senão se command inicia com "off_" então
12:                if command = "off_all" então
13:                    TURNOFFALLPOWERSUPPLIERSPORTS()
14:                senão
15:                    powerSupplyPort ← Extraia o número da porta após "off_"
16:                    TURNOFFPOWERSUPPLYPORT(powerSupplyPort)
17:            senão
18:                Informe "Comando inválido."
```

Os Algoritmos 3 e 4 são responsáveis, respectivamente, por energizar ou desenergizar todas as portas de fornecimento de energia e informar o resultado via serial.

Algoritmo 3 Função para energizar todas as portas

```
1: Função TURNONALLPOWERSUPPLIERSPORTS
2:     para cada pino em powerSupplyPins faça
3:         Configure pino como (LOW)     ▷ Envia sinal para energizar todas as portas
4:     Informe que todas as portas foram ligadas"
```

Algoritmo 4 Função para desenergizar todas as portas

```
1: Função TURNOFFALLPOWERSUPPLIERSPORTS
2:     para cada pino em powerSupplyPins faça
3:         Configure pino como (HIGH)     ▷ Envia sinal para desenergizar todas as portas
4:     Informe que todas as portas foram desligadas"
```

Os Algoritmos 5 e 6 são responsáveis, respectivamente, por ligar ou desligar uma porta de fornecimento de energia que é recebida por parâmetro e informar o resultado via serial.

Algoritmo 5 Função para energizar uma porta específica

```
1: Função TURNONPOWERSUPPLYPORT(powerSupplyPort)
2:     se ISVALIDPOWERSUPPLYPORT(powerSupplyPort) então
3:         Configure o pino da porta como (LOW) ▷ Envia sinal para energizar a porta
4:         Informe que a porta foi ligada
5:     senão
6:         Informe que a porta é inválida
```

Algoritmo 6 Função para desenergizar uma porta específica

```
1: Função TURNOFFPOWERSUPPLYPORT(powerSupplyPort)
2:     se ISVALIDPOWERSUPPLYPORT(powerSupplyPort) então
3:         Configure o pino da porta como (HIGH)   ▷ Envia sinal para desenergizar a porta
4:         Informe que a porta foi desligada
5:     senão
6:         Informe que a porta é inválida
```

O Algoritmo 7 tem a função de validar se a porta de fornecimento de energia é válida.

Algoritmo 7 Função para validar se a porta de fornecimento de energia é válida

```
1: Função IsValidPowerSupplyPort(powerSupplyPort)
2:     se powerSupplyPort ≥ 0 e powerSupplyPort < numPowerSuppliers então
3:         retorne True
4:     senão
5:         retorne False
```

3.3.2 Gerenciamento de energia de placas de prototipagem com fonte externa

Muitas placas de prototipagem com FPGA requerem uma potência superior à que uma única porta USB pode prover. Para atender a essa demanda, algumas placas vêm equipadas com um *jumper*, permitindo escolher entre alimentação via USB ou por uma fonte externa. Na Figura 3.5 é possível observar esse *jumper* na marcação com o nome *Power Select Jumper*.

Conforme discutido na Seção 3.3.1, cada placa pode ser alimentada individualmente por uma fonte externa. Para isso, basta apenas realizar a ligação elétrica correta no módulo relé. A figura 3.4 mostra um exemplo dessa ligação.

Figura 3.4: Esquema de ligação entre *Power Management Module* e fonte externa. Fonte: O autor. Fonte Placa de prototipagem: (Digilent, 2021b).

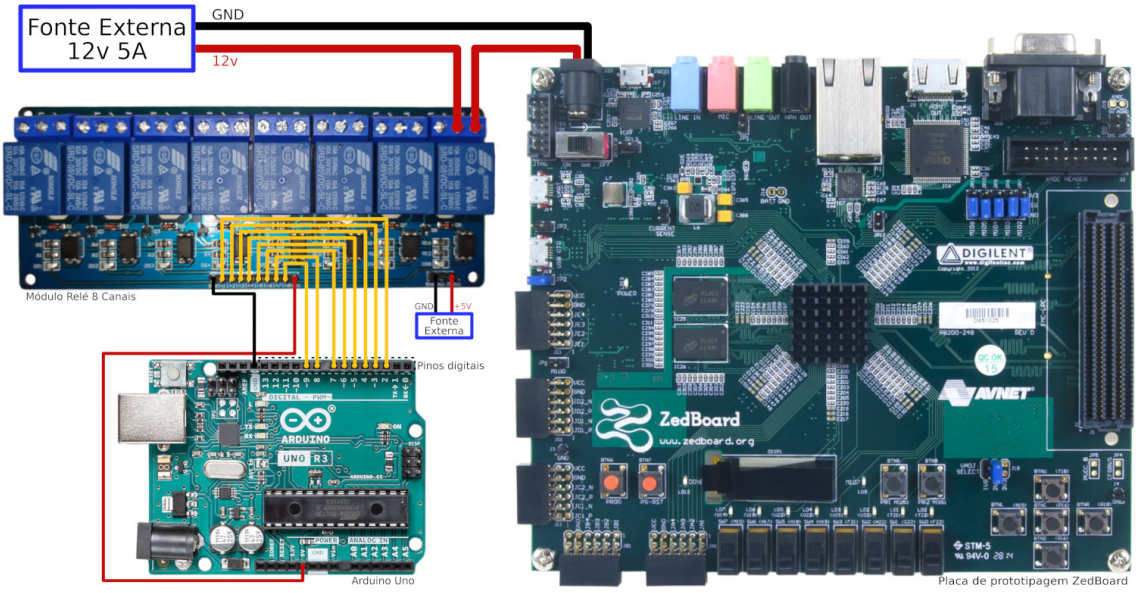
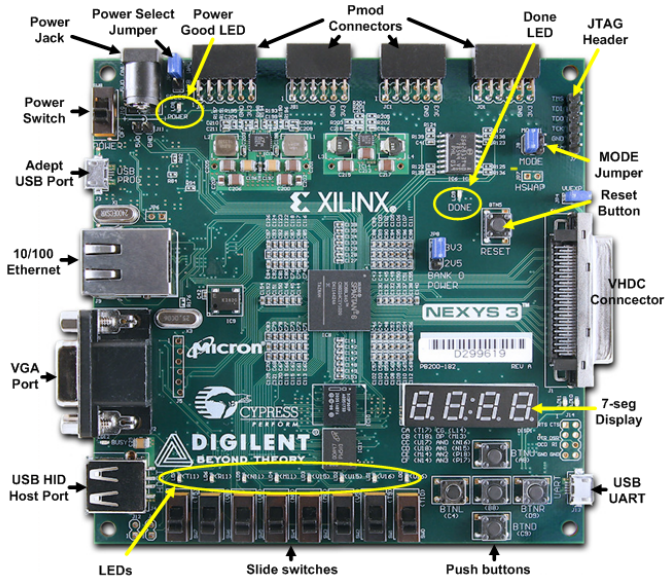


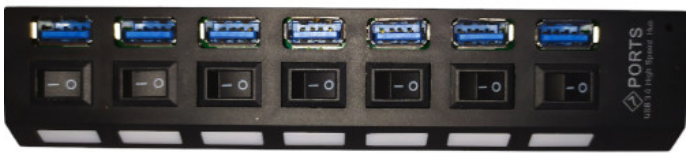
Figura 3.5: Placa de prototipagem Nexys 3. Fonte: (Digilent, 2021a).



3.3.3 Adaptação de Hub USB para o controle de energia

Um Hub USB com 7 portas (Figura 3.6) foi utilizado para permitir a comunicação e o fornecimento de energia para as placas de prototipagem com FPGA. Este Hub oferece a flexibilidade de ser alimentado tanto diretamente por uma porta USB quanto por uma fonte externa de 5V. Esse recurso é útil, visto que uma única entrada USB pode não prover energia suficiente para todas as 7 portas de saída.

Figura 3.6: Hub USB. Fonte: O autor.



Para permitir um controle individualizado da alimentação de cada porta do Hub através do *Power Management Module*, foi feita uma adaptação no dispositivo. Inicialmente, o Hub foi desmontado, conforme pode ser visto na Figura 3.7. Em seguida, foram soldados fios nos conetores de 5 volts de cada porta USB, confirme mostrado na Figura 3.8. Isso possibilitou o fornecimento de energia de forma individual para cada porta.

A Figura 3.9 mostra um esquema elétrico da ligação de uma porta USB. Neste esquema, observa-se que, quando o interruptor liga/desliga encontra-se na posição ‘ligado’, a porta USB recebe energia independentemente do *Power Management Module*. Por outro lado, se o interruptor estiver na posição ‘desligado’, a alimentação é gerenciada pelo

Power Management Module. Assim, para que o controle de energia ocorra de maneira automatizada, todos os interruptores do Hub devem permanecer na posição 'desligado'.

Figura 3.7: Hub USB desmontado. Fonte: O autor.

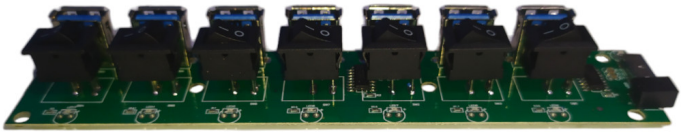


Figura 3.8: Fios soldados no Hub USB. Fonte: O autor.

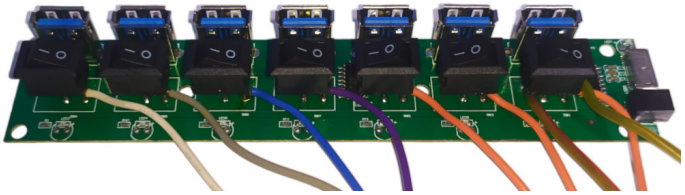
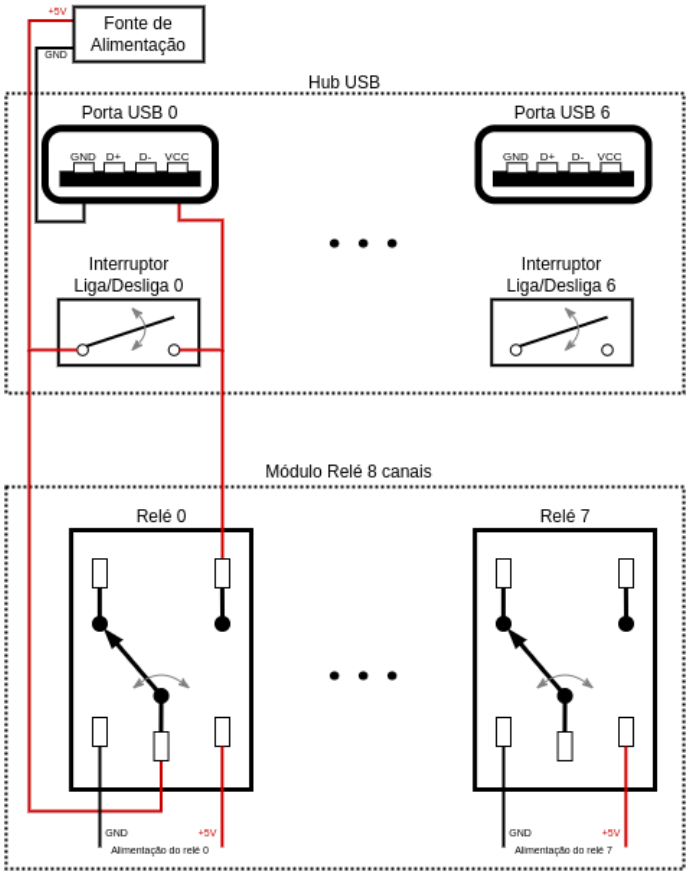
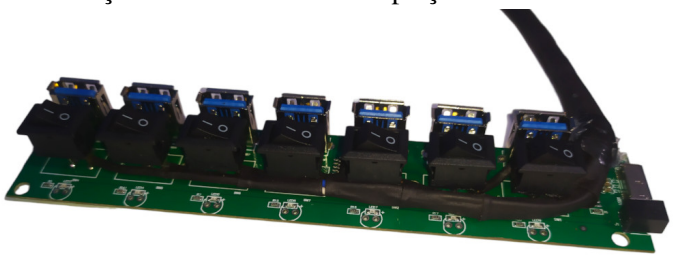


Figura 3.9: Esquema elétrico da ligação de uma porta USB. Fonte: O autor.



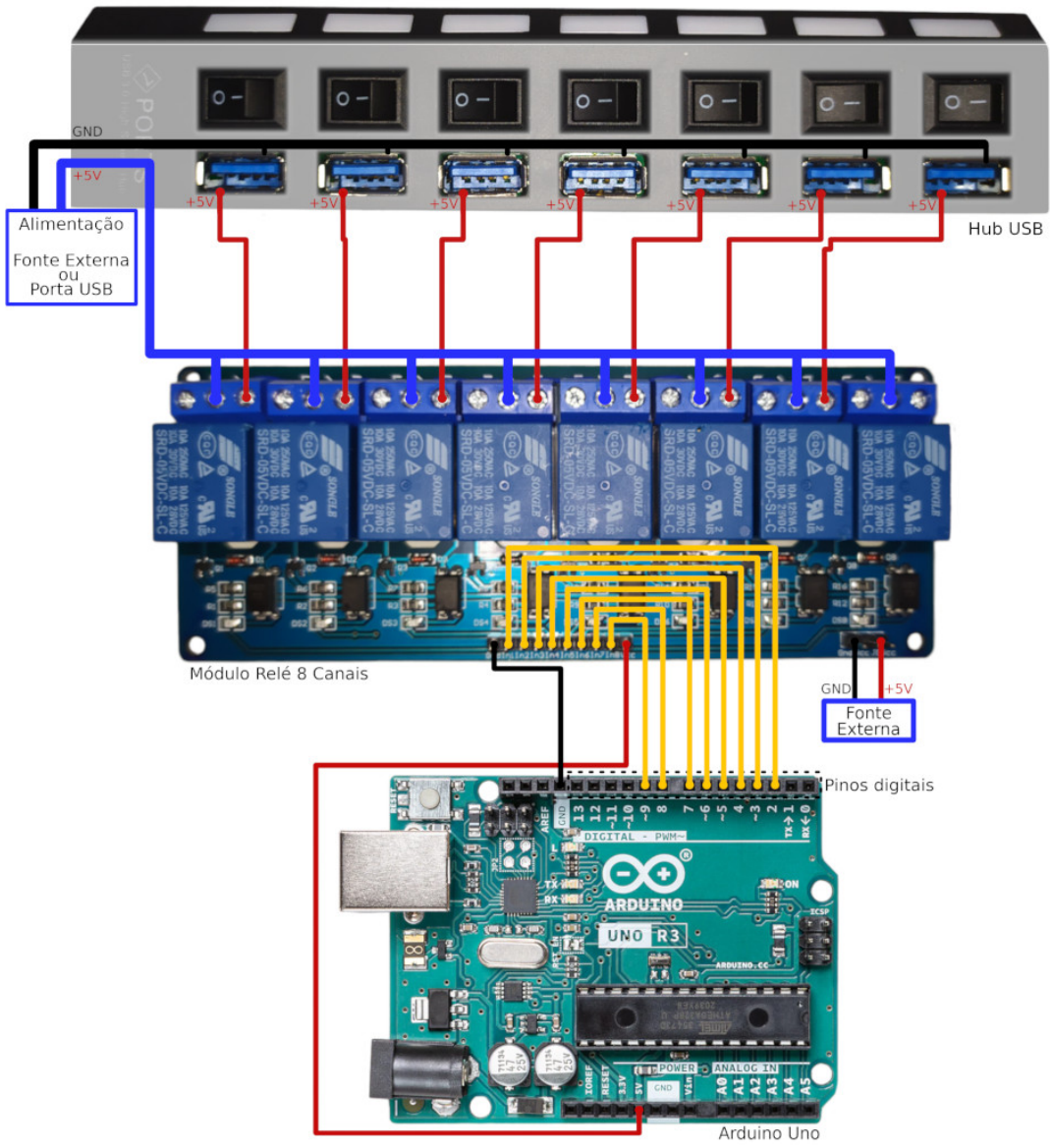
Finalmente, para um acabamento adequado, todos os fios foram agrupados e isolados utilizando fita termo retrátil, conforme mostrado na Figura 3.10.

Figura 3.10: Finalização e acabamento da adaptação do Hub USB. Fonte: O autor.



Após a adaptação do Hub, todos os fios de energia foram conectados ao módulo relé, conforme ilustrado na Figura 3.11.

Figura 3.11: Esquema de ligação entre Arduino, módulo relé e Hub USB. Fonte: O autor.



3.3.4 Construção da estrutura física

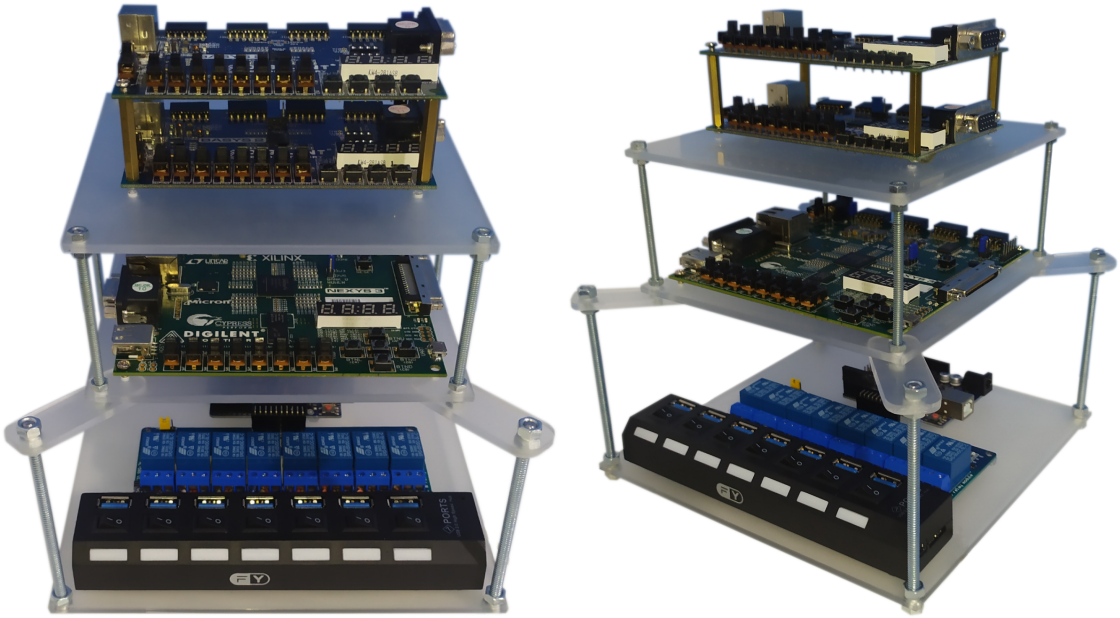
Com o intuito de conferir mais elegância ao projeto, foi elaborada uma estrutura modular para acomodar todos os equipamentos. Esta estrutura foi planejada em um design de múltiplos níveis, permitindo empilhamento de novos níveis e acomodação de uma maior quantidade de dispositivos.

Para a sua criação, foram utilizadas placas de acrílico, cortadas conforme as dimensões dos equipamentos. Foram feitas furações para garantir uma montagem correta e a fixação segura dos dispositivos. Posteriormente, o acrílico foi polido com uma lixa de grão 2000, resultando em um acabamento fosco e elegante.

No primeiro nível da estrutura, estão alocados os equipamentos responsáveis pelo gerenciamento de energia e conexão USB (Hub USB, módulo relé e Arduino Uno). Nos níveis subsequentes, foram posicionadas as placas de desenvolvimento FPGA: uma placa Digilent Nexys 3 no segundo nível e duas placas Digilent Basys 2 no terceiro.

O resultado desta montagem pode ser visto na figura 3.12.

Figura 3.12: Estrutura construída para conectar o sistema. Fonte: O autor.



(a) Visão frontal.

(b) Visão diagonal.

3.3.5 Serviço *FPGA-Loader*

O serviço *FPGA-Loader* é o responsável por programar uma dada FPGA. Para utilizar esse serviço, é necessário fazer uma requisição utilizando o método HTTP POST

para o *endpoint /upload* do serviço. Nessa requisição devem ser enviados, utilizando o formato *multipartform-data*, os dados:

- *file*: Arquivo *bitstream* que deve ser programado na FPGA.
- *serialNumber*: Número de série da FPGA que deve ser programada.
- *serialReadTime*: Tempo em segundos que deve ser feita a coleta de dados.
- *baudRate*: Velocidade de transmissão que os dados serão enviados para a porta serial (Esse valor é opcional. Caso ele não seja enviado, será definido o baudRate 115200 por padrão).

Quando o serviço *FPGA-Loader* recebe uma requisição direcionada ao *endpoint /upload*, a função *uploadFile* é acionada. No Algoritmo 8 alto nível e com detalhes o funcionamento dela.

Inicialmente a função recebe um arquivo *bitstream*, o número de série dda FPGA a ser programda, o tempo que deve ser feita a coleta de dados e a velocidade de transmissão da comunicação serial (caso informado). A função *getUSBSerialPorts* é chamada para obter a lista com todas as portas seriais conectadas no computador. Informações sobre a conexão física da FPGA e do conversor serial com o computador são obtidos através das funções *getFPGADataBySerialNumber* e *getPowerSupplyPorts*. Ao obter as portas de energia nas quais os dispositivos estão conectados, essas portas são energizadas através da função *turnOnTargetPowerSupply*. Em seguida, a função *identifyConnectedSerialPort* identifica a qual porta o dispositivo serial foi mapeado pelo sistema operacional, e uma *thread* é criada com a função *getFPGASerialData* para coletar os dados seriais.

Após esse processo, a FPGA é programada ao se chamar a função *uploadCode-ToFPGA*.

Quando a coleta de dados é concluída, a *thread* é encerrada. Os dados coletados por ela são salvos e as portas de energia previamente ativadas são desligadas através da função *turnOffTargetPowerSupply*. Por fim, os dados coletados são retornados como resposta à requisição.

Algoritmo 8 Função que processa a requisição para programar uma FPGA

1: **Função** UPLOADFILE
2: SALVE em disco o arquivo recebido na requisição POST
3: SALVE o serialNumber, serialReadTime e baudRate recebidos na requisição POST
4:
5: SALVE a lista de portas USB seriais conectadas no computador (função chamada: *getUSBSerialPorts*)
6:
7: SALVE os dados da FPGA a ser programada (função chamada: *getFPGADataBySerialNumber*)
8:
9: SALVE as portas de energia que a FPGA e o conversor serial estão conectados (função chamada: *getPowerSupplyPorts*)
10:
11: LIGUE a energia das portas-alvo (função chamada: *turnOnTargetPowerSupply*)
12: **se** erro ao ligar a energia das portas-alvo **então**
13: RETORNE erro com status 500
14:
15: IDENTIFIQUE a porta USB serial que foi conectada (função chamada: *identifyConnectedSerialPort*)
16: **se** erro ao identificar a porta serial USB que foi conectada **então**
17: RETORNE erro com status 500
18:
19: CRIE e INICIE uma thread para coletar dados da porta serial (função executada: *getFPGASerialData*)
20:
21: **tente**
22: PROGRAME a FPGA (função chamada: *uploadCodeToFPGA*)
23: SALVE a saída informada pelo software responsável por programar a FPGA
24: **exceção**
25: DESLIGUE a energia de todas as portas de energia (função chamada: *turnOnOffPowerSupply*)
26: RETORNE erro com status 500
27: —————— continua na próxima página ——————

28: AGUARDE a thread terminar
29: SALVE o resultado gerado na thread
30: DESLIGUE a energia das portas-alvo (função chamada: *turnOffTargetPowerSupply*)
31:
32: **se** erro ao desligar a energia das portas-alvo **então**
33: DESLIGUE a energia de todas as portas de energia (função chamada: *turnOnOff-PowerSupply*)
34: RETORNE erro com status 500
35:
36: **se** resultado da thread for vazio **então**
37: RETORNE erro com status 500
38:
39: RETORNE os dados coletados da serial (obtidos na thread)

O Algoritmo 9 faz uma busca por todas as portas seriais conectadas no sistema operacional, e retorna uma lista com todas as portas seriais que são do tipo USB.

Algoritmo 9 Função para obter as portas USB seriais conectadas

```
1: Função GETUSBSERIALPORTS
2:     allPorts ← Busque todas as portas seriais conectadas no computador
3:     usbPorts ← Crie uma lista vazia
4:     para port em allPorts faça
5:         se a palavra "USB"está contida em port.device então
6:             usbPorts ← Adicione port.device na lista
7:     retorne usbPorts
```

O Algoritmo 10 é responsável por obter informações sobre a conexão física da FPGA e do conversor serial com o computador. Para realizar essa busca, o serviço *DevMan* é acessado. Inicialmente é obtido por meio das variáveis de ambiente o *host* do serviço *DevMan* que foi definido no arquivo *.env*. Após é feita uma requisição utilizando o método HTTP GET para o *endpoint /devices/fpgas* enviando como parâmetro o número de série da FPGA que deseja-se buscar informações. Caso a requisição seja feita com sucesso, os dados são retornados.

Algoritmo 10 Função para obter informações da FPGA pelo Número Serial

1: **Função** GETFPGADATABYSERIALNUMBER(serialNumber)
2: $deviceManagerHost \leftarrow$ obter_Var_Amb("DEVICE_MANAGER_HOST")
3: $url \leftarrow$ construir_URL($deviceManagerHost$, serialNumber)
4: $response \leftarrow$ fazer_Requisição(url)
5: **se** $response.status\_code \neq 200$ **então**
6: **retorne** Erro: "Não foi possível obter dados da API"
7: $data \leftarrow$ converte_JSON($response$)
8: **retorne** $data$

O Algoritmo 11 coleta o número das portas fornecedoras de energia a partir dos dados que foram recebidos do serviço *DevMan*.

Algoritmo 11 Função para obter portas que fornecem energia a FPGA e ao conversor serial

1: **Função** GETPOWERSUPPLYPORTS($data$)
2: $ports \leftarrow$ **Cria** uma lista vazia
3: $fpgaPowerSupplyPort \leftarrow$ **Acesse** em $data$ o número da porta que fornece energia a FPGA
4: **Adicione** $fpgaPowerSupplyPort$ à lista $ports$
5: $serialCollectorPowerSupply \leftarrow$ **Acesse** em $data$ o dispositivo que fornece energia para o conversor serial
6: **se** $serialCollectorPowerSupply \neq$ `None` **então**
7: $serialCollectorPowerSupplyPort \leftarrow$ **Acesse** em $serialCollectorPowerSupply$ o número da porta
8: *Adicione* $serialCollectorPowerSupplyPort$ à lista $ports$
retorne $ports$

O Algoritmo 12 é responsável por receber uma lista de portas-alvo que devem ser energizadas (ou seja, as portas em que possuem dispositivos que serão utilizados) e chamar a função *turnOnOffPowerSupply* que será a responsável de fato por enviar o comando para que a porta seja energizada.

Algoritmo 12 Função para energizar as portas-alvo

```
1: Função TURNONTARGETPOWERSUPPLY(ports, startup_time)
2:     delay ← 0
3:     para index, port em enumerate(ports) faça
4:         se index == (tamanho de ports - 1) então
5:             delay ← startup_time
6:         err ← turnOnOffPowerSupply("on", port, delay)
7:         se err então
8:             retorne "Não foi possível ligar a USB " + port
       retorne 0
```

O Algoritmo 13 executa uma operação de diferença entre a lista atual de portas seriais e uma lista anterior (antes das portas serem energizadas). Com isso, é possível obter com precisão qual é a porta serial em que os dados devem ser coletados.

Algoritmo 13 Função para identificar a porta serial que for conectada

```
1: Função IDENTIFYCONNECTEDSERIALPORT(usbSerialPorts)
2:     connectedSerialPort ← getPowerSupplyPorts() - usbSerialPorts
3:     err ← None
4:     se tamanho de connectedSerialPort ≠ 1 então
5:         se tamanho de connectedSerialPort = 0 então
6:             err ← "Nenhuma porta serial foi conectada. Tente novamente"
7:         senão
8:             err ← "Muitas portas seriais conectadas. Não foi possível identificar a
       porta que os dados devem ser capturados. Tente novamente"
9:         turnOnOffPowerSupply("off", "all", 0)
10:        retorne None, err
11:    connectedSerialPort ← connectedSerialPort[0]
12:    retorne connectedSerialPort, err
```

O Algoritmo 14 é responsável por coletar os dados gerados em tempo real pela FPGA. Para isso, é utilizado o serviço *FPGA Serial Data Collector (FSerDaC)*.

Inicialmente, é obtido por meio das variáveis de ambiente o *host* do serviço *FPGA Serial Data Collector (FSerDaC)* que foi definido no arquivo *.env*. Para iniciar a coleta de dados, uma requisição utilizando o método HTTP POST é enviada para o *endpoint /get_fpga_serial_data* utilizando o formato *multipartform-data*. São enviadas nessa re-

quisição a porta serial, o tempo que os dados devem ser coletados e a velocidade de transmissão. Caso a requisição seja feita com sucesso, os dados são retornados.

Algoritmo 14 Função para coletar dados serial gerados pela FPGA

```
1:  Função GETFPGASERIALDATA(serialPort, serialReadTime, baudRate, q)
2:      data ← {'serialPort': serialPort, 'serialReadTime': serialReadTime}
3:      se baudRate ≠ None então
4:          data['baudRate'] ← baudRate
5:      tente
6:          fserdacHost ← obter_Var_Amb("FSERDAC_HOST")
7:          url ← "http://fserdacHost:8000/get_fpga_serial_data"
8:          response ← Faz POST para url enviando data
9:          se response.status_code = 200 então
10:             q.inserir(response.text)
11:             retorne
12:         senão
13:             q.inserir(None)
14:             retorne
15:     exceção
16:         q.inserir(None)
17:         retorne
```

O Algoritmo 15 é responsável por receber uma lista de portas-alvo que devem ser desenergizadas (ou seja, as portas em que possuem dispositivos que já foram utilizado e já podem ser desligados) e chamar a função *turnOnOffPowerSupply* que será a responsável de fato por enviar o comando para que a porta seja desenergizada.

Algoritmo 15 Função para desligar a energia das portas-alvo

```
1:  Função TURNOFFTARGETPOWERSUPPLY(ports)
2:      delay ← 0
3:      para cada port em ports faça
4:          err ← turnOnOffPowerSupply("off", port, delay)
5:          se err então retorne "Não foi possível desligar a USB " +
    port
        retorne 0
```

O Algoritmo 16 é responsável por programar a FPGA. Para isso, o *script* 3.3.11

é executado enviando o nome do arquivo *bitstream* e o número de série da FPGA a ser programada.

Algoritmo 16 Função para fazer o upload do bitstream para a FPGA

```
1: Função UPLOADCODETOFPGA(filename, serialNumber)
2:     se serialNumber = None então
3:         raise Retorna mensagem de erro
4:     tente
5:         output ← Executa script para programar a FPGA. filename e serialNumber são enviados como parâmetro
6:     exceção
7:         raise Retorna mensagem de erro
8:     retorne output
```

O Algoritmo 17 é responsável por energizar ou desenergizar uma porta de fornecimento de energia. Para isso, essa função se comunica com o serviço *Remote Power Supply Control (RPSCon)*.

Inicialmente, é obtido por meio das variáveis de ambiente o *host* do serviço *Remote Power Supply Control (RPSCon)* que foi definido no arquivo *.env*. Para enviar o comando, uma requisição utilizando o método HTTP POST é enviada para o *endpoint /power_supply_control* utilizando o formato *multipartform-data*. São enviadas nessa requisição a ação que deve ser executada, o número da porta de fornecimento de energia que a ação deve ser executada e o tempo que deve ser aguardado após a última porta ser energizada. Caso a requisição seja feita com sucesso, são retornados os dados fornecidos pelo serviço.

Algoritmo 17 Função para ligar ou desligar a energia de portas especificadas

1: **Função** TURNONOFFPOWERSUPPLY($action, powerSupplyPort, timeSleep = 0$)
2: $data \leftarrow$ 'action': $action$, 'powerSupplyPort': $powerSupplyPort$, 'timeSleep': $timeSleep$
3: **tente**
4: $rpsconHost \leftarrow$ obter_Var_Amb("RPSCON_HOST")
5: $url \leftarrow$ "http://$rpsconHost$:8000/power_supply_control"
6: $response \leftarrow$ Faz POST para url enviando $data$
7: **se** $response.status\_code = 200$ **então**
8: **retorne** 0
9: **senão**
10: **retorne** 1
11: **exceção**
12: **retorne** 1

3.3.6 Serviço *FPGA Serial Data Collector* (FSerDaC)

O serviço FSerDaC é o responsável por coletar em tempo real os dados recebidos na porta serial do computador. Para utilizar esse serviço, é necessário fazer uma requisição utilizando o método HTTP POST para o *endpoint /get_fpga_serial_data* do serviço. Nessa requisição devem ser enviados, utilizando o formato *multipartform-data*, os valores:

- *serialPort*: Porta serial USB que o conversor serial está conectado (Ex.: *ttyUSB0, ttyUSB1 etc.*).
- *serialReadTime*: Tempo em segundos que deve ser feita a coleta de dados (Ex.: *20*).
- *baudRate*: Velocidade de transmissão dos dados recebidos (Esse valor é opcional. Caso ele não seja enviado, será definifo o baudRate 115200 por padrão).

Quando o serviço *FSerDaC* recebe uma requisição direcionada para o *endpoint* /get_fpga_serial_data, a função *getFPGASerialData* é chamada. Essa função recebe os valores mencionados via POST, cria uma conexão serial com o dispositivo conectado em *serialPort* e com a taxa de transmissão definida em *baudRate*, coleta os dados enviados por esse dispositivo durante o tempo definido em *serialReadTime*, salva em um arquivo e retorna os dados desse arquivo.

O funcionamento dessa função pode ser visto com detalhes no Algoritmo 18.

Algoritmo 18 Função para coletar dados da FPGA via porta serial

```
1:  BAUD_RATE_DEFAULT ← 115200
2:  Função GETFPGASERIALDATA
3:      serialPort, serialReadTime, baudRate ← valores recebidos via POST
4:      se baudRate = None então
5:          baudRate ← BAUD_RATE_DEFAULT
6:      senão
7:          baudRate ← Converte para inteiro(baudRate)
8:
9:      se serialPort = None or serialReadTime = None então
10:         retorne "Erro: serialPort e o serialReadTime devem ser enviados"
11:
12:     serialPort ← converte para string (serialPort)
13:     serialReadTime ← converte para inteiro(serialReadTime)
14:     tente
15:         ser ← inicia a conexão serial em serialPort com baudRate
16:     exceção
17:         retorne "Não foi possível conectar com a porta serial serialPort"
18:
19:     fileName ← 'collectedData.txt'
20:     tente
21:         startTime ← time.now()
22:         enquanto time.now() - startTime < serialReadTime faça
23:             serialData ← ser.readline()
24:             decodedData ← serialData.decode().strip()
25:             escrever decodedData no arquivo fileName
26:     exceção
27:         retorne "Houve um erro ao ler os dados da porta serial"
28:     por fim
29:         fecha a conexão serial de ser
30:
31:     retorne conteúdo do arquivo fileName
```

3.3.7 Serviço *Remote Power Supply Control* (RPSCon)

O serviço *RPSCon* é o responsável por realizar a conexão com o Módulo de Gerenciamento de Energia (3.3.1) e enviar comandos para ligar ou desligar a energia das portas de fornecimento de energia. Para utilizar esse serviço, é necessário fazer uma requisição utilizando o método HTTP POST para o *endpoint /power_supply_control* do serviço. Nessa requisição devem ser enviados, utilizando o formato *multipartform-data*, os dados:

- *action*: Ação a ser realizada da porta. Para ligar deve ser enviado *on*. Para desligar deve ser enviado *off*
- *powerSupplyPort*: Número da porta em que deve ser executada uma ação. Pode ser enviado um número inteiro ou *all* para executar a ação em todas as portas.
- *timeSleep*: Tempo em segundos que o serviço deve aguardar após executar a ação.

Quando o serviço *RPSCon* é iniciado são feitas algumas configurações. Inicialmente, é obtido através das variáveis de ambiente o *idVendor* e *idProduct* do Arduino que foram informados no arquivo *.env*. Com essas informações, inicia-se uma busca pela porta serial em que o Arduino está conectado (é possível ver isso em detalhes no Algoritmo 20).

Quando a porta serial é adquirida, uma conexão é estabelecida com o Arduino, iniciando assim a comunicação serial. Após isso, o serviço se torna acessível, permitindo receber requisições e interagir com o Arduino. A lógica desta parte pode ser vista no Algoritmo 19.

Algoritmo 19 Função de inicialização do serviço RPSCon

1: $idVendor \leftarrow$ obter variável de ambiente ("ID_VENDOR_ARDUINO")
2: $idProduct \leftarrow$ obter variável de ambiente ("ID_PRODUCT_ARDUINO")
3: $serialPort \leftarrow$ procura a porta USB que o arduino com $idVendor$ e $idProduct$ está conectado (função chamada: *searchUSBDdevice*)
4: **se** $serialPort \neq$`None` **então**
5: $\quad baudRate \leftarrow 9600$
6: $\quad ser \leftarrow$ inicia a conexão serial em $serialPort$ com $baudRate$
7: $\quad$ aguarde 2 segundos para a inicialização do Arduino
8: $\quad$ inicie o serviço
9: **senão**
10: $\quad$ encerrar a execução com erro

Algoritmo 20 Função para buscar porta em que um dispositivo USB está conectado

```
1: Função SEARCHUSBDDEVICE(idVendor, idProduct)
2:     para cada port em lista de todas as portas seriais conectadas faça
3:         se idVendor = port.vid e idProduct = port.pid então
4:             retorne port.device
5:     retorne None
```

Quando o serviço *RPSCon* recebe uma requisição direcionada ao *endpoint /power_supply_control*, a função *powerSupplyControl* é chamada. Essa função é responsável obter os dados enviados via POST, chamar a função *sendCommand()* (a lógica pode ser vista no Algoritmo 22) para enviar o comando, obter e retornar a resposta do Arduino. No Algoritmo 21 é mostrado com detalhes o funcionamento dela.

Algoritmo 21 Função para receber requisições e se comunicar com o *Power Management Module* (3.3.1)

```
1: Função POWERSUPPLYCONTROL
2:     action, powerSupplyPort, timeSleep ← Receba valores via POST
3:     tente
4:         SENDCOMMAND(action_powerSupplyPort)
5:         arduinoResponse ← Receba a resposta do Arduino
6:         Aguarde o tempo definido em timeSleep
7:     exceção
8:         response ← Crie uma resposta de erro com o código 500
9:         retorne response
10:    retorne arduinoResponse
```

Algoritmo 22 Função para enviar um comando para o *Power Management Module* (3.3.1)

```
1: Função POWERSUPPLYCONTROL(command)
2:     tente
3:         Envie command codificado para a conexão serial
4:     exceção
5:         Crie uma exceção informando o erro
```

3.3.8 Serviço *Real Time Video Capture* (ReTiCap)

O serviço *ReTiCap* é o responsável por transmitir imagens em tempo real. Para utilizar esse serviço, é necessário fazer uma requisição utilizando o método HTTP GET para o *endpoint* **/** do serviço.

Quando o serviço *ReTiCap* recebe uma requisição direcionada ao *endpoint* **/**, a função *getRealTimeVideo* e *getVideo* são chamadas. Nos Algoritmos 23 e 24 são mostrados com detalhes o funcionamento delas.

Algoritmo 23 Função para transmitir vídeo em tempo real

```
1: Função GETREALTIMEVIDEO
2:     retorne vídeo em tempo real obtido pela função getVideo()
```

Algoritmo 24 Função para obter vídeo da webcam e transmitir como MJPEG

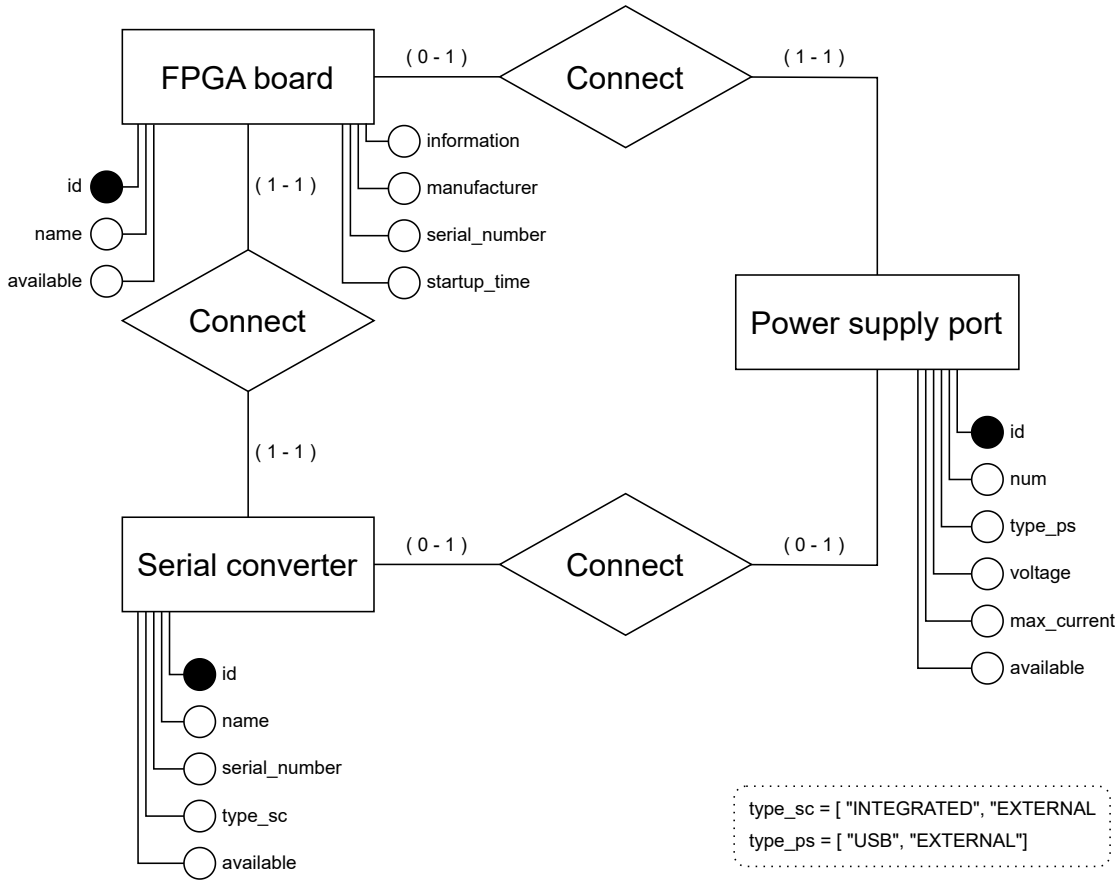
```
1: Função GETVIDEO
2:     cap ← inicia a câmera principal do computador
3:     enquanto True faça                ▷ Lê continuamente cada frame da câmera.
4:         ret, frame ← lê frame da câmera
5:         se not ret então              ▷ Verificar se o frame foi lido com sucesso.
6:             Parar
7:
8:         ret, buffer ← converte frame para o formato JPEG
9:         frame ← converte imagem em bytes para a transmissão
10:        transmite como MJPEG
```

3.3.9 Serviço *SPIDER - Device Manager* (DevMan)

Administradores de laboratórios necessitam informar ao sistema os dispositivos disponíveis para serem utilizados remotamente. O serviço *SPIDER - Device Manager* foi planejado com este propósito. Com ele, torna-se possível cadastrar placas de prototipagem com FPGA, conversores de dados serial, e portas de fornecimento de energia.

Inicialmente foi criado um diagrama de entidade relacionamento. Nele foram definidas as entidades, relacionamentos e atributos de relevância. O diagrama completo pode ser visto na Figura 3.13.

Figura 3.13: Diagrama entidade relacionamento do serviço *SPIDER - Device Manager*. Fonte: O autor.



Conversores de dados serial podem ser externos, como as placas adaptadoras que utilizam o *chip* CP2102 (Silicon Labs, 2023), ou podem estar integrados em placas de prototipagem que já contêm o conversor embutido. Por esse motivo, a entidade '*Serial converter*' possui o atributo 'type_sc' que aceita somente os valores '*INTEGRATED*' ou '*EXTERNAL*'

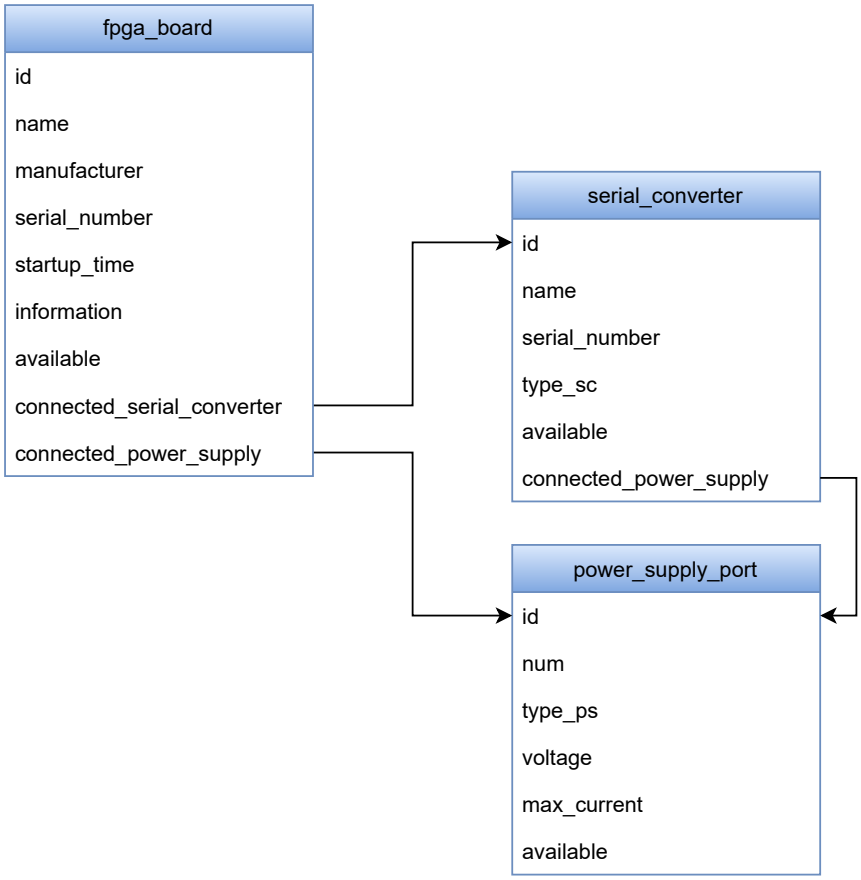
Portas de fornecimento de energia podem prover energia por meio de uma conexão USB ou a partir de uma fonte externa, conforme mencionado, respectivamente, nas seções (3.3.3) e (3.3.2). Por esse motivo, a entidade '*Power supply*' possui o atributo 'type_ps' que aceita somente os valores 'USB' ou '*EXTERNAL*'.

O atributo '*information*' na entidade 'FPGA *board*' possibilita ao administrador adicionar informações específicas sobre as placas, como o pino destinado à transmissão de dados seriais, a frequência e o pino '*clock*', entre outras particularidades.

Na modelagem definida na Figura 3.13, cada placa com FPGA deve estar vinculada tanto a um conversor serial quanto a uma porta de fornecimento de energia. Já o conversor serial pode estar conectado a uma única porta de fornecimento de energia ou

não estar conectado a nenhuma. Isso acontece porque algumas placas já possuem um conversor serial integrado. Assim, ao energizar a placa, o conversor integrado também é automaticamente energizado. Após a etapa de modelagem, as tabelas e suas respectivas relações foram criadas, conforme apresentado na Figura 3.14.

Figura 3.14: Tabelas do banco de dados do serviço *SPIDER-Device Manager*. Fonte: O autor.

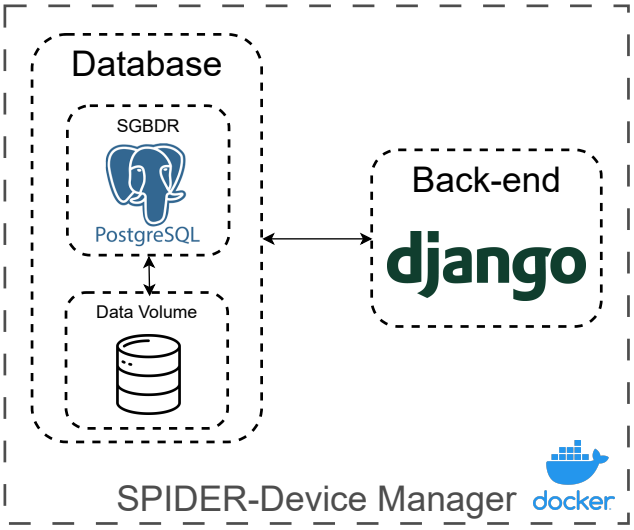


O serviço *SPIDER - Device Manager* foi desenvolvido com o poderoso Framework Django (3.1.1). Além de utilizar Django, foi criada uma API dinâmica e versátil utilizando o Django REST *Framework* (3.1.2). Ela permite uma interação ágil e eficaz com os dados do serviço.

Pensando na segurança e eficiência do armazenamento dos dados, foi escolhido o PostgreSQL (3.1.5), um renomado sistema de banco de dados relacional de código aberto, conhecido pela sua robustez e confiabilidade. Para garantir que o serviço seja consistente e facilmente escalável, foi utilizado Docker (3.1.6) e Docker Compose (3.1.7) para provisionar a infraestrutura. Isso nos permite conteinerizar a aplicação, garantindo que ela opere de forma uniforme em diferentes ambientes.

A Figura 3.15 mostra a arquitetura do serviço.

Figura 3.15: Arquitetura do serviço *SPIDER-Device Manager*. Fonte: O autor.



Essa ferramenta é essencial para permitir que o Mapa de inventário (2.3) e o Serviço ´FPGALoader' (3.3.5) obtenham dados atualizados das placas, fazendo com que o usuário tenha uma melhor experiência e o sistema seja mais automatizado.

Através desta integração, o sistema pode determinar com precisão quais portas fornecedoras de energia devem ser ativadas conforme a placa deseja-se programar.

3.3.10 Serviço *SPIDER - Access*

Esse serviço tem o objetivo de receber requisições externas e, de acordo com *end-point* acessado, redirecionar para o serviço apropriado. Para isso, o Nginx (3.1.4) foi utilizado e configurado para atuar como um *proxy* reverso, assegurando um redirecionamento eficaz e transparente para os destinos corretos.

3.3.11 *Script* FPGA - *Uploader*

Para que a FPGA seja programada, foi criado um *script* na linguagem *bash*, que recebe o nome de um arquivo *bitstream* e o número de série de uma FPGA que deve ser programada. Após, o programa Digilent djtgcfg é executado enviando os parâmetros necessários para a programação da FPGA. O Algoritmo 25 mostra o funcionamento do *script*.

Algoritmo 25 *Script* para programar uma FPGA

1: $nameFile, serialNumber \leftarrow$ **Receba** por parâmetro nameFile e serialNumber
2: **Execute** o comando "djtgcfg prog -d SN:$serialNumber$ -i 0 -f ./$nameFile$"

3.3.12 Integração do sistema SPIDER com o Mapa de Inventário

O Mapa de Inventário (2.3) foi escolhido para ser utilizado como um *front-end* para o sistema SPIDER. Essa escolha se deu ao fato de ele ser uma ferramenta muito poderosa para representar dados de diversos tipos.

A proposta de exibir os dispositivos em um mapa é proporcionar aos usuários clareza sobre a localização exata onde seus experimentos estão sendo executados, tornando a experiência um pouco mais 'física'. Do mesmo modo, ao fornecer transmissões ao vivo das placas, busca-se fazer com que o usuário tenha uma sensação mais próxima de um laboratório real.

Além disso, ao adotar o mapa de inventário, laboratórios interessados podem registrar placas para uso presencial, satisfazendo a demanda de usuários que valorizam a interação direta com as placas físicas.

Inicialmente, foi feita uma modelagem dos dados de placas de prototipagem equipadas com FPGA para possibilitar que os dados fossem exibidos da melhor maneira no mapa. Após, iniciou-se o desenvolvimento de uma interface para a programação remota das placas.

3.3.12.1 Modelagem dos dados para o Mapa de inventário

Para realizar a modelagem, observou-se que o sistema de mapas possui três elementos básicos: *Menus*, *Tags* e *Locations*. Ao acessar o site das principais fabricantes de placas de prototipagem com FPGA, foi possível determinar quatro atributos-chave: 'Fabricante', 'Família de FPGA', 'Modelo' e 'Tipo de acesso'. Esses atributos foram mapeados para o elemento *Menu*.

Para facilitar o entendimento, a Figura 3.16 mostra um exemplo simples de mapeamento.

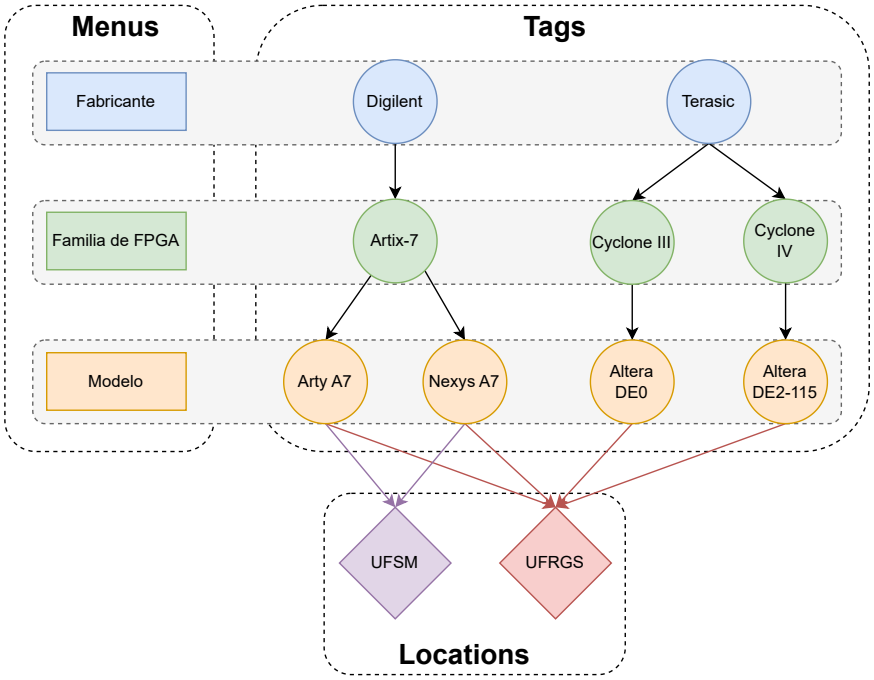
Para cada *Menu*, foram adicionados elementos relacionados ao nome do dele. Por exemplo, no *Menu* 'Fabricante', foram adicionados os elementos 'Digilent' e 'Terasic'. Esses elementos são mapeados para *Tags*.

Locais que possuem placas foram mapeados para *Location*. Por exemplo, ‘UFSM’ e ‘UFRGS’

Após isso, foram criadas relações de *Tags* para *Location*, permitindo informar ao sistema quais locais as *tags* estão presentes em determinadas localizações. Conforme a Figura 3.16, as *Tags* ‘Arty A7’ e ‘Nexys A7’ estão contidas, por exemplo, na localização UFSM e UFRGS.

Também foram criadas relações de *Tags* com *Tags*, fazendo com que existam relações de hierarquia. Por exemplo, uma *Tag* chamada ‘Terasic’ pode ter as *Tags* ‘Cyclone III’ e ‘Cyclone IV’ como filhas.

Figura 3.16: Diagrama de mapeamento de dados para o Mapa de Inventário. Fonte: O autor.



3.3.12.2 Desenvolvimento da interface de usuário para programação e coleta de dados

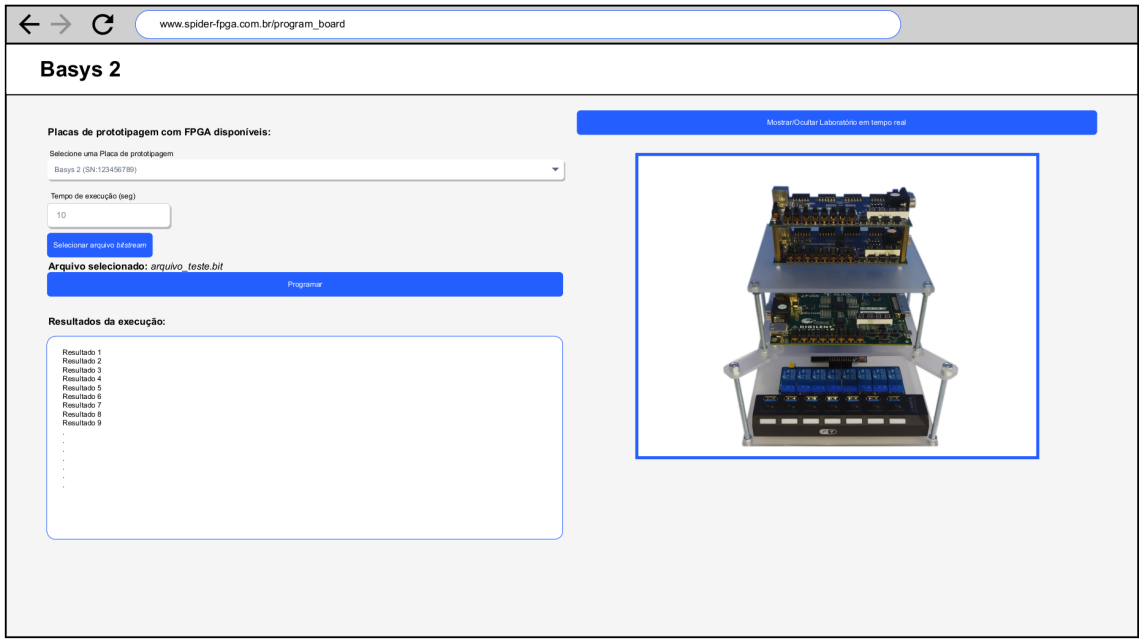
Visando criar uma tela intuitiva e simples, foi inicialmente criado um ‘mockup’ de uma tela. Como pode ser visto na Figura 3.17, foram posicionados:

- *Dropdown* de seleção de placa: Permite escolher uma placa específica pelo número de série, especialmente útil quando há múltiplas placas do mesmo modelo no laboratório. Essa informação é coletada do Serviço *SPIDER-Device Manager* (3.3.9).
- Campo de tempo: Permite a inserção do tempo em segundos que os dados devem ser coletados.
- Seletor de arquivo *bitstream*: Permite que um arquivo *bitstream* seja selecionado

para ser utilizado para programar a placa.

- Botão ‘Programar’: Inicia a programação e coleta de dados da placa ao enviar uma solicitação à infraestrutura do sistema SPIDER. Esse botão só fica habilitado após os itens anteriores serem devidamente definidos.
- Caixa saída de dados: Exibe os dados coletados quando eles estão disponíveis. Permanece oculto na ausência de dados.
- Botão para exibir/ocultar imagens do laboratório: Permite exibir ou ocultar imagens em tempo real do laboratório.
- Caixa de *stream* de vídeo: Exibe imagens em tempo real do laboratório.

Figura 3.17: Mockup da interface de usuário para programação e coleta de dados. Fonte: O autor.

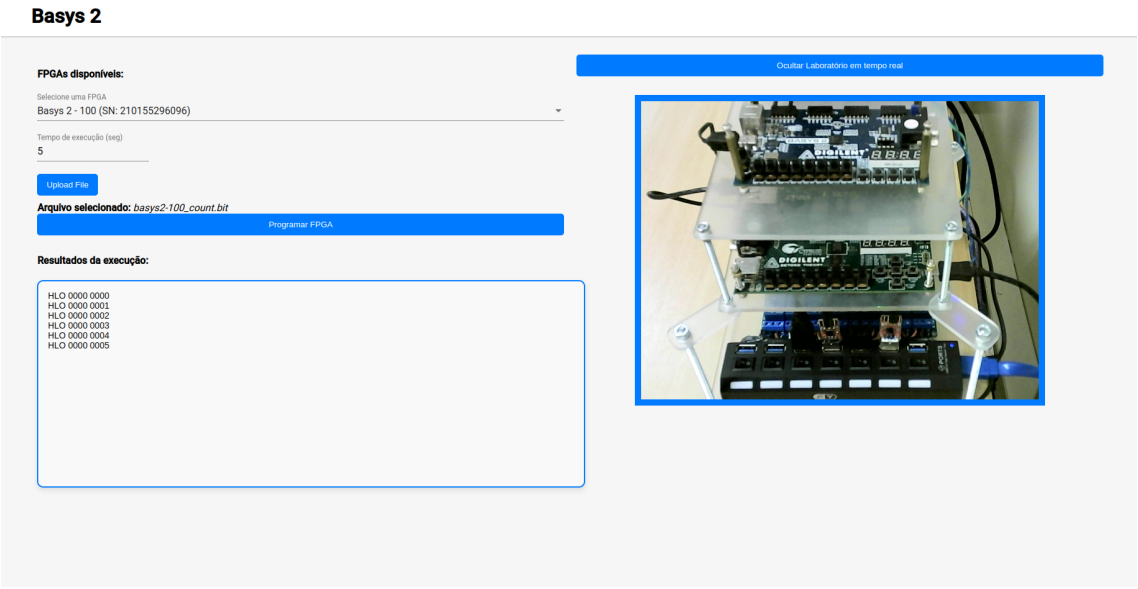


Esse *mockup* foi convertido em uma tela dentro do Mapa de Inventário (2.3). Essa tela foi desenvolvida utilizando Angular (3.1.3). Foram criados funções para garantir o comportamento adequado dos componentes da tela e foram configurados os *endpoints* do sistema SPIDER necessários para o correto funcionamento.

O *framework* Angular (3.1.3) foi utilizado para criar uma interface de usuário integrada ao Mapa de Inventário (2.3). Funções foram implementadas e *endpoints* do sistema SPIDER foram configurados para garantir o funcionamento correto do *front-end*

A Figura 3.18 mostra o resultado do desenvolvimento dessa interface de usuário.

Figura 3.18: Interface de usuário para programação e coleta de dados. Fonte: O autor.



3.3.13 Fluxo completo de funcionamento

Após a detalhada explicação da infraestrutura do SPIDER e sua integração com o *front-end*, é fundamental oferecer um resumo do fluxo completo para facilitar o entendimento. Com essa finalidade, foi criado um diagrama de sequência que pode ser visualizado na Figura 3.19.

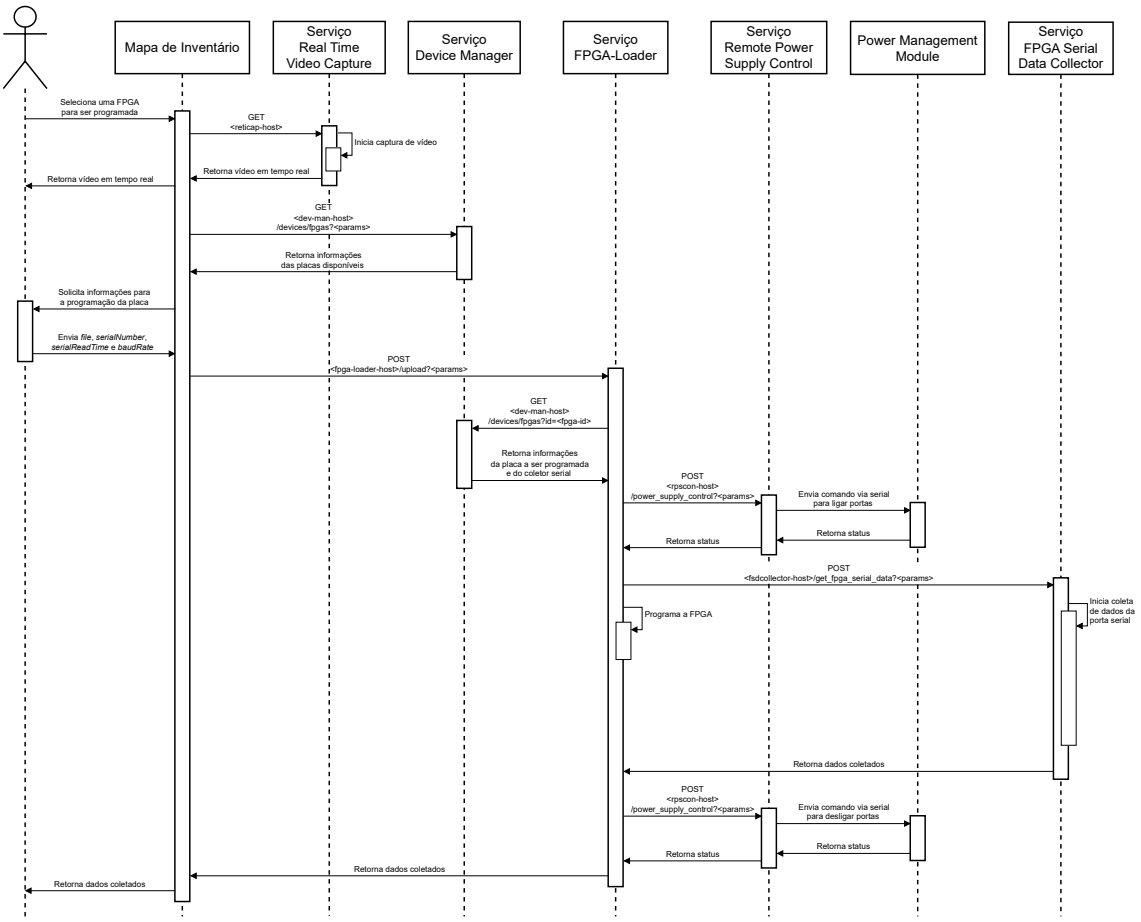
Pode-se observar inicialmente um usuário selecionando no Mapa de Inventário uma placa com FPGA para ser programada. É feita uma requisição para o serviço *SPIDER-Access*, que solicita imagens em tempo real para o serviço *ReTiCap*. Simultaneamente, informações sobre as placas com FPGA disponíveis são requisitadas ao serviço *Device Manager*.

Nesse momento o usuário seleciona a placa a ser programada, juntamente com o arquivo *bitstream*, o tempo de coleta de dados e velocidade de transmissão dos dados serial. O Mapa de Inventário envia a requisição ao serviço *SPIDER-Access* que redireciona para o serviço *FPGA-Loader*, este solicita informações ao serviço *Device Manager* e envia uma solicitação para o serviço *RPSCon* ativar o fornecimento de energia das portas que os dispositivos-alvo estão conectados. O serviço *RPSCon* envia comando para o módulo *PoMaM*, que por sua vez ativa fisicamente o fornecimento de energia das portas.

Após isso, o serviço *FPGA-Loader* envia uma solicitação para o serviço *FSerDaC*, que inicia a coleta de dados seriais. Em paralelo a isso, o *FPGA-Loader* programa a placa de prototipagem. Quando os dados tiverem sido coletados pelo tempo informado pelo

usuário, os dados são retornados ao usuário, por meio do Mapa de Inventário.

Figura 3.19: Diagrama de sequência do sistema SPIDER. Fonte: O autor.



4 RESULTADOS

4.1 Acesso às APIs dos serviços

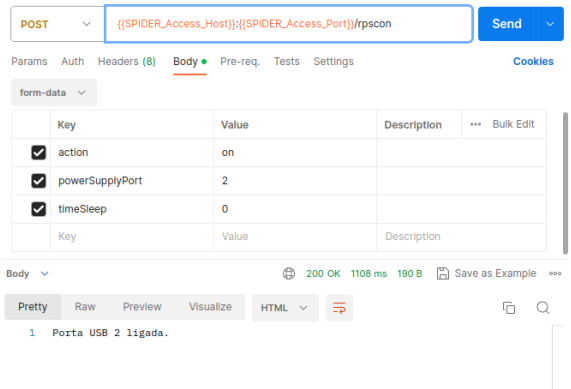
Nessa seção, será mostrado o funcionamento das APIs criadas. Para realizar as requisições será utilizado o *Postman* (POSTMAN, 2023).

4.1.1 *Remote Power Supply Control*

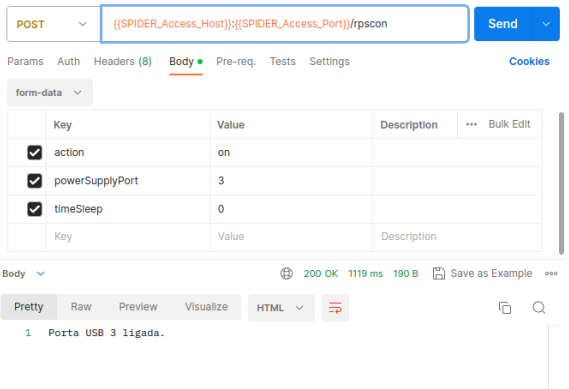
Iniciou-se testando o serviço *RPSCon*. Serão feitas algumas requisições e serão observados os resultados.

Nas Figuras 4.1a e 4.1b, são enviadas requisições para ligar o fornecimento de energia das portas USB 2 e 3.

Figura 4.1: Realizando requisição para o fornecimento de energia. Fonte: O autor.



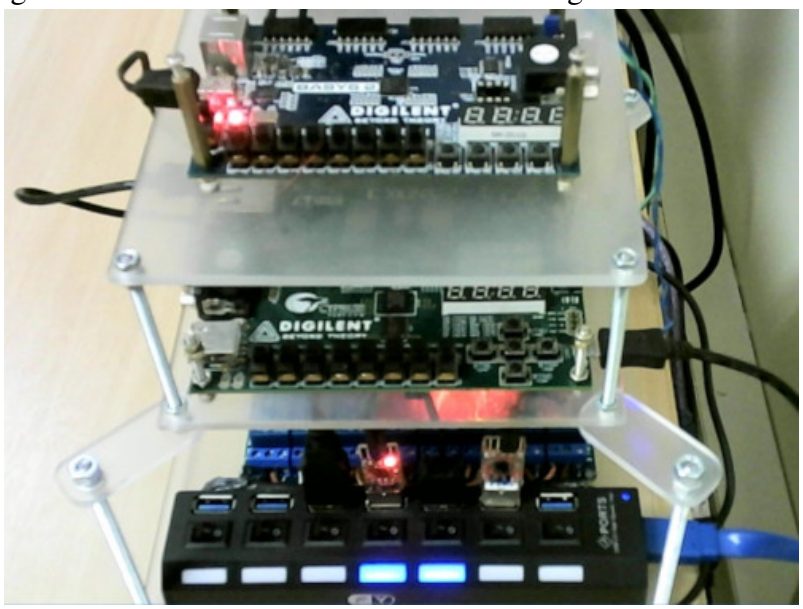
(a) Serviço *RPSCon* ligando o fornecimento de energia da porta USB 2.



(b) Serviço *RPSCon* ligando o fornecimento de energia da porta USB 3.

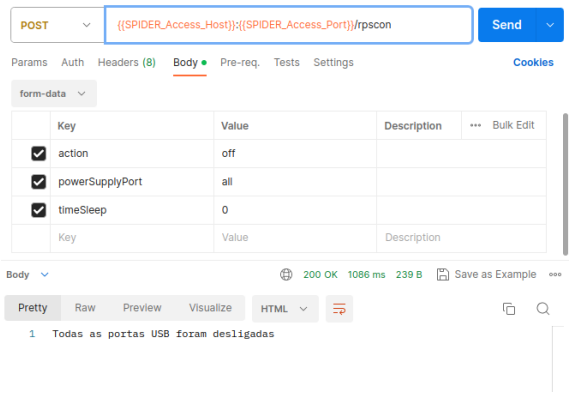
Após as requisições serem realizadas, pode-se visualizar na transmissão de vídeo que as portas USB 2 e 3 foram devidamente ligadas (Figura 4.2).

Figura 4.2: Portas USB 2 e 3 fornecendo energia. Fonte: O autor.

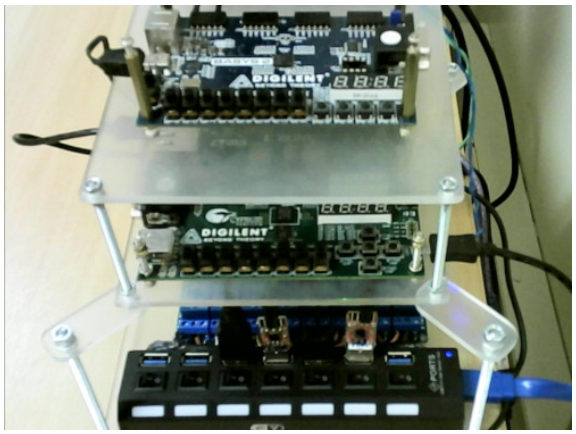


Uma requisição será enviada para desligar o fornecimento de energia de todas as portas (Figura 4.3a). Na Figura 4.3b, é possível observar o resultado da execução.

Figura 4.3: Serviço *RPSCon* desligando o fornecimento de energia de todas as portas USB. Fonte: O autor.



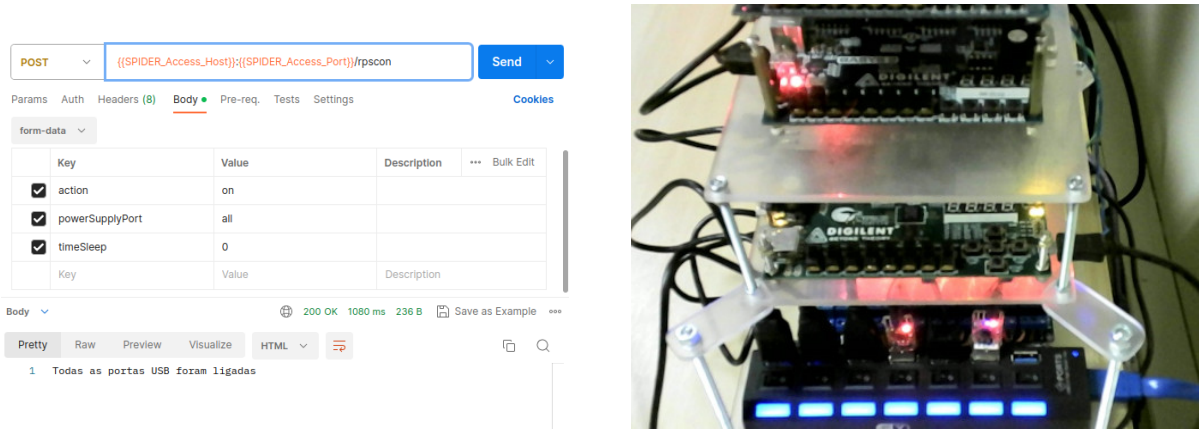
(a) Requisição sendo enviada.



(b) Resultado após a execução.

Neste teste, foi enviada uma requisição para ligar o fornecimento de energia de todas as portas (Figura 4.4a). Na Figura 4.4b, é possível verificar o resultado dessa execução.

Figura 4.4: Serviço *RPSCon* ligando o fornecimento de energia de todas as portas USB. Fonte: O autor.



(a) Requisição sendo enviada.

(b) Resultado após a execução.

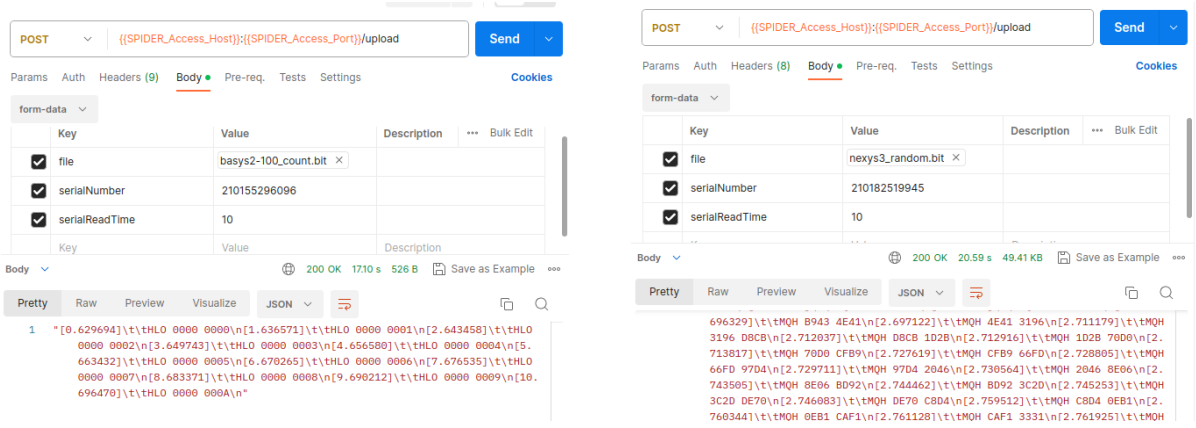
4.1.2 FPGA-Loader

Para testar a API do Serviço *FPGA-Loader*, foram feitas requisições utilizando o método HTTP POST enviando os valores ‘file’, ‘serialNumber’ e ‘serialReadTime’. Como a taxa de transmissão usada é o valor padrão (115200), o valor baudRate não foi enviado.

Para a placa Basys 2, foi enviado um programa que descreve um contador. Ele incrementa o valor um a cada segundo. Já para a placa Basys 3, foi enviado um programa que descreve gerador de números aleatórios, a partir de uma ‘semente’.

A Figura 4.5a mostra os resultados de execução da Basys 2 e a Figura 4.5b mostra os resultados da Nexys 3.

Figura 4.5: Resultados de execução obtidos pelos Serviço *FPGA-Loader*. Fonte: O autor.



(a) Dados de execução de programa na Basys 2.

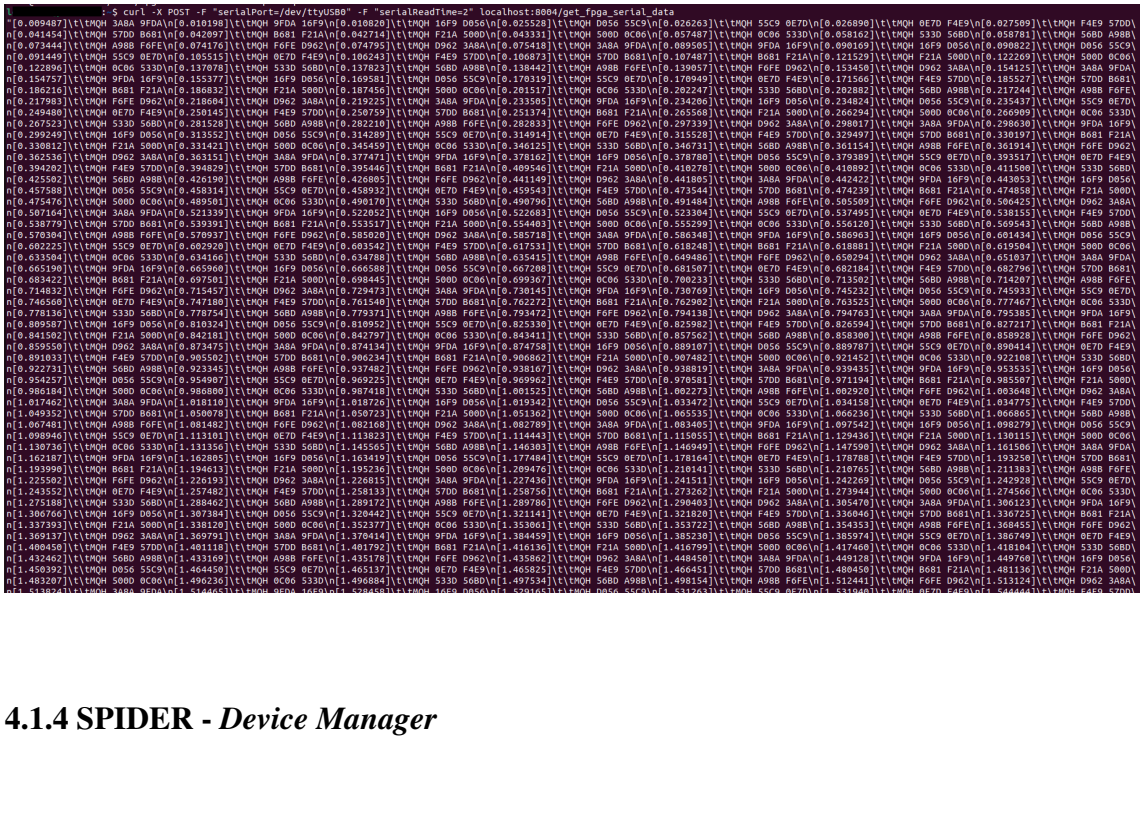
(b) Dados de execução de programa na Nexys 3.

4.1.3 FPGA *Serial Data Collector*

Para realizar o teste do serviço *FSerDaC* individualmente e coletar dados, sem o restante do sistema, foi necessário fazer um procedimento manual, já que não seria possível coletar os dados caso o canal serial já estivesse em uso. Dessa forma, o acesso a API foi feita de forma isolada. Para isso, com privilégios de um usuário administrativo, foi acessado o servidor onde a infraestrutura é executada. Após, foi acessado o contêiner responsável por programar as placas e foi feito o *upload* de um programa manualmente, evitando que o serviço de coleta de dados fosse chamado automaticamente. Isso impediria o teste de ser feito, já que o canal serial ficaria ocupado.

Com a placa programada e gerando dados, foi feita uma requisição para o serviço *FSerDaC* solicitando coletar os dados do dispositivo serial conectado em '/dev/ttyUSB0' durante 2 segundos. A resposta desta requisição pode ser vista na Figura 4.6.

Figura 4.6: Dados coletados pelo serviço *FSerDaC*. Fonte: O autor.



$ curl -X POST -F "serialPort=/dev/ttyUSB0" -F "serialReadTime=2" localhost:8004/get_fpga_serial_data
"[0.009487]\t\tMQH 3A8A 9FDA\n[0.010198]\t\tMQH 9FDA 16F9\n[0.010820]\t\tMQH 16F9 D056\n[0.025528]\t\tMQH D056 55C9\n[0.026263]\t\tMQH 55C9 0E7D\n[0.026890]\t\tMQH 0E7D F4E9\n[0.027509]\t\tMQH F4E9 57DD\n[0.041454]\t\tMQH 57DD B681\n[0.042097]\t\tMQH B681 F21A\n[0.042714]\t\tMQH F21A 500D\n[0.043331]\t\tMQH 500D 0C06\n[0.057487]\t\tMQH 0C06 533D\n[0.058162]\t\tMQH 533D 56BD\n[0.058781]\t\tMQH 56BD A98B\n[0.073444]\t\tMQH A98B F6FE\n[0.074176]\t\tMQH F6FE D962\n[0.074795]\t\tMQH D962 3A8A\n[0.075418]\t\tMQH 3A8A 9FDA\n[0.089505]\t\tMQH 9FDA 16F9\n[0.090169]\t\tMQH 16F9 D056\n[0.090822]\t\tMQH D056 55C9\n[0.091449]\t\tMQH 55C9 0E7D\n[0.105515]\t\tMQH 0E7D F4E9\n[0.106243]\t\tMQH F4E9 57DD\n[0.106873]\t\tMQH 57DD B681\n[0.107487]\t\tMQH B681 F21A\n[0.121529]\t\tMQH F21A 500D\n[0.122269]\t\tMQH 500D 0C06\n[0.122896]\t\tMQH 0C06 533D\n[0.137078]\t\tMQH 533D 56BD\n[0.137823]\t\tMQH 56BD A98B\n[0.138442]\t\tMQH A98B F6FE\n[0.139057]\t\tMQH F6FE D962\n[0.153450]\t\tMQH D962 3A8A\n[0.154125]\t\tMQH 3A8A 9FDA\n[0.154757]\t\tMQH 9FDA 16F9\n[0.155377]\t\tMQH 16F9 D056\n[0.169581]\t\tMQH D056 55C9\n[0.170319]\t\tMQH 55C9 0E7D\n[0.170949]\t\tMQH 0E7D F4E9\n[0.171566]\t\tMQH F4E9 57DD\n[0.185527]\t\tMQH 57DD B681\n[0.186216]\t\tMQH B681 F21A\n[0.186832]\t\tMQH F21A 500D\n[0.187456]\t\tMQH 500D 0C06\n[0.201517]\t\tMQH 0C06 533D\n[0.202247]\t\tMQH 533D 56BD\n[0.202882]\t\tMQH 56BD A98B\n[0.217244]\t\tMQH A98B F6FE\n[0.217983]\t\tMQH F6FE D962\n[0.218604]\t\tMQH D962 3A8A\n[0.219225]\t\tMQH 3A8A 9FDA\n[0.233505]\t\tMQH 9FDA 16F9\n[0.234206]\t\tMQH 16F9 D056\n[0.234824]\t\tMQH D056 55C9\n[0.235437]\t\tMQH 55C9 0E7D\n[0.249480]\t\tMQH 0E7D F4E9\n[0.250145]\t\tMQH F4E9 57DD\n[0.250759]\t\tMQH 57DD B681\n[0.251374]\t\tMQH B681 F21A\n[0.265568]\t\tMQH F21A 500D\n[0.266294]\t\tMQH 500D 0C06\n[0.266909]\t\tMQH 0C06 533D\n[0.267523]\t\tMQH 533D 56BD\n[0.281528]\t\tMQH 56BD A98B\n[0.282210]\t\tMQH A98B F6FE\n[0.282833]\t\tMQH F6FE D962\n[0.297339]\t\tMQH D962 3A8A\n[0.298017]\t\tMQH 3A8A 9FDA\n[0.298630]\t\tMQH 9FDA 16F9\n[0.299249]\t\tMQH 16F9 D056\n[0.313532]\t\tMQH D056 55C9\n[0.314289]\t\tMQH 55C9 0E7D\n[0.314914]\t\tMQH 0E7D F4E9\n[0.315528]\t\tMQH F4E9 57DD\n[0.329497]\t\tMQH 57DD B681\n[0.330197]\t\tMQH B681 F21A\n[0.330812]\t\tMQH F21A 500D\n[0.331421]\t\tMQH 500D 0C06\n[0.345459]\t\tMQH 0C06 533D\n[0.346125]\t\tMQH 533D 56BD\n[0.346731]\t\tMQH 56BD A98B\n[0.361154]\t\tMQH A98B F6FE\n[0.361914]\t\tMQH F6FE D962\n[0.362536]\t\tMQH D962 3A8A\n[0.363151]\t\tMQH 3A8A 9FDA\n[0.377471]\t\tMQH 9FDA 16F9\n[0.378162]\t\tMQH 16F9 D056\n[0.378780]\t\tMQH D056 55C9\n[0.379389]\t\tMQH 55C9 0E7D\n[0.393517]\t\tMQH 0E7D F4E9\n[0.394202]\t\tMQH F4E9 57DD\n[0.394829]\t\tMQH 57DD B681\n[0.395446]\t\tMQH B681 F21A\n[0.409546]\t\tMQH F21A 500D\n[0.410278]\t\tMQH 500D 0C06\n[0.410892]\t\tMQH 0C06 533D\n[0.411500]\t\tMQH 533D 56BD\n[0.425502]\t\tMQH 56BD A98B\n[0.426190]\t\tMQH A98B F6FE\n[0.426805]\t\tMQH F6FE D962\n[0.441149]\t\tMQH D962 3A8A\n[0.441805]\t\tMQH 3A8A 9FDA\n[0.442422]\t\tMQH 9FDA 16F9\n[0.443053]\t\tMQH 16F9 D056\n[0.457588]\t\tMQH D056 55C9\n[0.458314]\t\tMQH 55C9 0E7D\n[0.458932]\t\tMQH 0E7D F4E9\n[0.459543]\t\tMQH F4E9 57DD\n[0.473544]\t\tMQH 57DD B681\n[0.474239]\t\tMQH B681 F21A\n[0.474858]\t\tMQH F21A 500D\n[0.475476]\t\tMQH 500D 0C06\n[0.489501]\t\tMQH 0C06 533D\n[0.490170]\t\tMQH 533D 56BD\n[0.490796]\t\tMQH 56BD A98B\n[0.491484]\t\tMQH A98B F6FE\n[0.505509]\t\tMQH F6FE D962\n[0.506425]\t\tMQH D962 3A8A\n[0.507164]\t\tMQH 3A8A 9FDA\n[0.521339]\t\tMQH 9FDA 16F9\n[0.522052]\t\tMQH 16F9 D056\n[0.522683]\t\tMQH D056 55C9\n[0.523304]\t\tMQH 55C9 0E7D\n[0.537495]\t\tMQH 0E7D F4E9\n[0.538155]\t\tMQH F4E9 57DD\n[0.538779]\t\tMQH 57DD B681\n[0.539391]\t\tMQH B681 F21A\n[0.553517]\t\tMQH F21A 500D\n[0.554403]\t\tMQH 500D 0C06\n[0.555299]\t\tMQH 0C06 533D\n[0.556120]\t\tMQH 533D 56BD\n[0.569543]\t\tMQH 56BD A98B\n[0.570304]\t\tMQH A98B F6FE\n[0.570937]\t\tMQH F6FE D962\n[0.585028]\t\tMQH D962 3A8A\n[0.585718]\t\tMQH 3A8A 9FDA\n[0.586348]\t\tMQH 9FDA 16F9\n[0.586963]\t\tMQH 16F9 D056\n[0.601434]\t\tMQH D056 55C9\n[0.602225]\t\tMQH 55C9 0E7D\n[0.602920]\t\tMQH 0E7D F4E9\n[0.603542]\t\tMQH F4E9 57DD\n[0.617531]\t\tMQH 57DD B681\n[0.618248]\t\tMQH B681 F21A\n[0.618881]\t\tMQH F21A 500D\n[0.619504]\t\tMQH 500D 0C06\n[0.633504]\t\tMQH 0C06 533D\n[0.634166]\t\tMQH 533D 56BD\n[0.634788]\t\tMQH 56BD A98B\n[0.635415]\t\tMQH A98B F6FE\n[0.649486]\t\tMQH F6FE D962\n[0.650294]\t\tMQH D962 3A8A\n[0.651037]\t\tMQH 3A8A 9FDA\n[0.665190]\t\tMQH 9FDA 16F9\n[0.665960]\t\tMQH 16F9 D056\n[0.666588]\t\tMQH D056 55C9\n[0.667208]\t\tMQH 55C9 0E7D\n[0.681507]\t\tMQH 0E7D F4E9\n[0.682184]\t\tMQH F4E9 57DD\n[0.682796]\t\tMQH 57DD B681\n[0.683422]\t\tMQH B681 F21A\n[0.697501]\t\tMQH F21A 500D\n[0.698445]\t\tMQH 500D 0C06\n[0.699367]\t\tMQH 0C06 533D\n[0.700233]\t\tMQH 533D 56BD\n[0.713502]\t\tMQH 56BD A98B\n[0.714207]\t\tMQH A98B F6FE\n[0.714832]\t\tMQH F6FE D962\n[0.715457]\t\tMQH D962 3A8A\n[0.729473]\t\tMQH 3A8A 9FDA\n[0.730145]\t\tMQH 9FDA 16F9\n[0.730769]\t\tMQH 16F9 D056\n[0.745232]\t\tMQH D056 55C9\n[0.745933]\t\tMQH 55C9 0E7D\n[0.746560]\t\tMQH 0E7D F4E9\n[0.747180]\t\tMQH F4E9 57DD\n[0.761540]\t\tMQH 57DD B681\n[0.762272]\t\tMQH B681 F21A\n[0.762902]\t\tMQH F21A 500D\n[0.763525]\t\tMQH 500D 0C06\n[0.777467]\t\tMQH 0C06 533D\n[0.778136]\t\tMQH 533D 56BD\n[0.778754]\t\tMQH 56BD A98B\n[0.779371]\t\tMQH A98B F6FE\n[0.793472]\t\tMQH F6FE D962\n[0.794138]\t\tMQH D962 3A8A\n[0.794763]\t\tMQH 3A8A 9FDA\n[0.795385]\t\tMQH 9FDA 16F9\n[0.809587]\t\tMQH 16F9 D056\n[0.810324]\t\tMQH D056 55C9\n[0.810952]\t\tMQH 55C9 0E7D\n[0.825330]\t\tMQH 0E7D F4E9\n[0.825982]\t\tMQH F4E9 57DD\n[0.826594]\t\tMQH 57DD B681\n[0.827217]\t\tMQH B681 F21A\n[0.841502]\t\tMQH F21A 500D\n[0.842181]\t\tMQH 500D 0C06\n[0.842797]\t\tMQH 0C06 533D\n[0.843411]\t\tMQH 533D 56BD\n[0.857562]\t\tMQH 56BD A98B\n[0.858300]\t\tMQH A98B F6FE\n[0.858928]\t\tMQH F6FE D962\n[0.859550]\t\tMQH D962 3A8A\n[0.873475]\t\tMQH 3A8A 9FDA\n[0.874134]\t\tMQH 9FDA 16F9\n[0.874758]\t\tMQH 16F9 D056\n[0.889107]\t\tMQH D056 55C9\n[0.889767]\t\tMQH 55C9 0E7D\n[0.890414]\t\tMQH 0E7D F4E9\n[0.891033]\t\tMQH F4E9 57DD\n[0.905502]\t\tMQH 57DD B681\n[0.906234]\t\tMQH B681 F21A\n[0.906862]\t\tMQH F21A 500D\n[0.907482]\t\tMQH 500D 0C06\n[0.921452]\t\tMQH 0C06 533D\n[0.922108]\t\tMQH 533D 56BD\n[0.922731]\t\tMQH 56BD A98B\n[0.923345]\t\tMQH A98B F6FE\n[0.937482]\t\tMQH F6FE D962\n[0.938167]\t\tMQH D962 3A8A\n[0.938819]\t\tMQH 3A8A 9FDA\n[0.939435]\t\tMQH 9FDA 16F9\n[0.953535]\t\tMQH 16F9 D056\n[0.954257]\t\tMQH D056 55C9\n[0.954907]\t\tMQH 55C9 0E7D\n[0.969225]\t\tMQH 0E7D F4E9\n[0.969962]\t\tMQH F4E9 57DD\n[0.970581]\t\tMQH 57DD B681\n[0.971194]\t\tMQH B681 F21A\n[0.985507]\t\tMQH F21A 500D\n[0.986184]\t\tMQH 500D 0C06\n[0.986800]\t\tMQH 0C06 533D\n[0.987418]\t\tMQH 533D 56BD\n[1.001525]\t\tMQH 56BD A98B\n[1.002273]\t\tMQH A98B F6FE\n[1.002920]\t\tMQH F6FE D962\n[1.003648]\t\tMQH D962 3A8A\n[1.017462]\t\tMQH 3A8A 9FDA\n[1.018110]\t\tMQH 9FDA 16F9\n[1.018726]\t\tMQH 16F9 D056\n[1.019342]\t\tMQH D056 55C9\n[1.033472]\t\tMQH 55C9 0E7D\n[1.034158]\t\tMQH 0E7D F4E9\n[1.034775]\t\tMQH F4E9 57DD\n[1.049352]\t\tMQH 57DD B681\n[1.050078]\t\tMQH B681 F21A\n[1.050723]\t\tMQH F21A 500D\n[1.051362]\t\tMQH 500D 0C06\n[1.065535]\t\tMQH 0C06 533D\n[1.066236]\t\tMQH 533D 56BD\n[1.066865]\t\tMQH 56BD A98B\n[1.067481]\t\tMQH A98B F6FE\n[1.081482]\t\tMQH F6FE D962\n[1.082168]\t\tMQH D962 3A8A\n[1.082789]\t\tMQH 3A8A 9FDA\n[1.083405]\t\tMQH 9FDA 16F9\n[1.097542]\t\tMQH 16F9 D056\n[1.098279]\t\tMQH D056 55C9\n[1.098946]\t\tMQH 55C9 0E7D\n[1.113101]\t\tMQH 0E7D F4E9\n[1.113823]\t\tMQH F4E9 57DD\n[1.114443]\t\tMQH 57DD B681\n[1.115055]\t\tMQH B681 F21A\n[1.129436]\t\tMQH F21A 500D\n[1.130115]\t\tMQH 500D 0C06\n[1.130736]\t\tMQH 0C06 533D\n[1.131356]\t\tMQH 533D 56BD\n[1.145565]\t\tMQH 56BD A98B\n[1.146303]\t\tMQH A98B F6FE\n[1.146949]\t\tMQH F6FE D962\n[1.147590]\t\tMQH D962 3A8A\n[1.161506]\t\tMQH 3A8A 9FDA\n[1.162187]\t\tMQH 9FDA 16F9\n[1.163419]\t\tMQH 16F9 D056\n[1.163995]\t\tMQH D056 55C9\n[1.177484]\t\tMQH 55C9 0E7D\n[1.178164]\t\tMQH 0E7D F4E9\n[1.178788]\t\tMQH F4E9 57DD\n[1.193258]\t\tMQH 57DD B681\n[1.193990]\t\tMQH B681 F21A\n[1.194613]\t\tMQH F21A 500D\n[1.195236]\t\tMQH 500D 0C06\n[1.209476]\t\tMQH 0C06 533D\n[1.210141]\t\tMQH 533D 56BD\n[1.210765]\t\tMQH 56BD A98B\n[1.211383]\t\tMQH A98B F6FE\n[1.225502]\t\tMQH F6FE D962\n[1.226193]\t\tMQH D962 3A8A\n[1.226815]\t\tMQH 3A8A 9FDA\n[1.227436]\t\tMQH 9FDA 16F9\n[1.241511]\t\tMQH 16F9 D056\n[1.242269]\t\tMQH D056 55C9\n[1.242928]\t\tMQH 55C9 0E7D\n[1.243552]\t\tMQH 0E7D F4E9\n[1.257482]\t\tMQH F4E9 57DD\n[1.258133]\t\tMQH 57DD B681\n[1.258756]\t\tMQH B681 F21A\n[1.273262]\t\tMQH F21A 500D\n[1.273944]\t\tMQH 500D 0C06\n[1.274566]\t\tMQH 0C06 533D\n[1.275188]\t\tMQH 533D 56BD\n[1.288462]\t\tMQH 56BD A98B\n[1.289172]\t\tMQH A98B F6FE\n[1.289786]\t\tMQH F6FE D962\n[1.290403]\t\tMQH D962 3A8A\n[1.305470]\t\tMQH 3A8A 9FDA\n[1.306126]\t\tMQH 9FDA 16F9\n[1.307384]\t\tMQH 16F9 D056\n[1.320442]\t\tMQH D056 55C9\n[1.321141]\t\tMQH 55C9 0E7D\n[1.321828]\t\tMQH 0E7D F4E9\n[1.321828]\t\tMQH F4E9 57DD\n[1.336046]\t\tMQH 57DD B681\n[1.336725]\t\tMQH B681 F21A\n[1.337393]\t\tMQH F21A 500D\n[1.338120]\t\tMQH 500D 0C06\n[1.352377]\t\tMQH 0C06 533D\n[1.353061]\t\tMQH 533D 56BD\n[1.353722]\t\tMQH 56BD A98B\n[1.354353]\t\tMQH A98B F6FE\n[1.368455]\t\tMQH F6FE D962\n[1.369137]\t\tMQH D962 3A8A\n[1.369791]\t\tMQH 3A8A 9FDA\n[1.370414]\t\tMQH 9FDA 16F9\n[1.384459]\t\tMQH 16F9 D056\n[1.385230]\t\tMQH D056 55C9\n[1.385974]\t\tMQH 55C9 0E7D\n[1.386749]\t\tMQH 0E7D F4E9\n[1.400450]\t\tMQH F4E9 57DD\n[1.401118]\t\tMQH 57DD B681\n[1.401792]\t\tMQH B681 F21A\n[1.416136]\t\tMQH F21A 500D\n[1.416799]\t\tMQH 500D 0C06\n[1.417460]\t\tMQH 0C06 533D\n[1.418104]\t\tMQH 533D 56BD\n[1.432462]\t\tMQH 56BD A98B\n[1.433169]\t\tMQH A98B F6FE\n[1.435178]\t\tMQH F6FE D962\n[1.435862]\t\tMQH D962 3A8A\n[1.448450]\t\tMQH 3A8A 9FDA\n[1.449128]\t\tMQH 9FDA 16F9\n[1.449760]\t\tMQH 16F9 D056\n[1.450392]\t\tMQH D056 55C9\n[1.464458]\t\tMQH 55C9 0E7D\n[1.465137]\t\tMQH 0E7D F4E9\n[1.465825]\t\tMQH F4E9 57DD\n[1.466451]\t\tMQH 57DD B681\n[1.480450]\t\tMQH B681 F21A\n[1.481136]\t\tMQH F21A 500D\n[1.483207]\t\tMQH 500D 0C06\n[1.496236]\t\tMQH 0C06 533D\n[1.496884]\t\tMQH 533D 56BD\n[1.497534]\t\tMQH 56BD A98B\n[1.498154]\t\tMQH A98B F6FE\n[1.512441]\t\tMQH F6FE D962\n[1.513124]\t\tMQH D962 3A8A\n

4.1.4 SPIDER - *Device Manager*

Para mostrar o funcionamento da API do Serviço *SPIDER-Device Manager*, foram cadastrados alguns dados no sistema. Na Seção 4.2 será mostrado como realizar o cadastro.

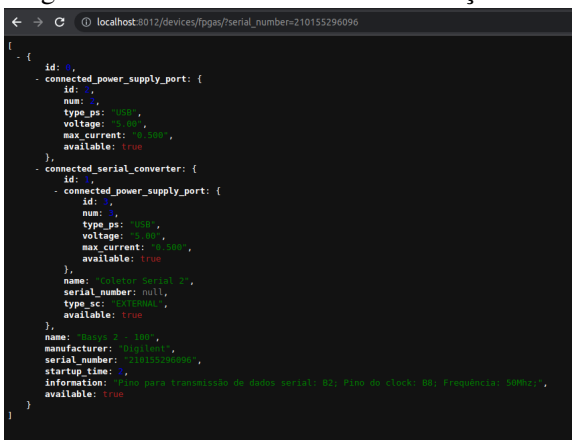
Ao acessar o *endpoint* '/devices/fpgas/', recebe-se informações de todas as placas juntamente com conversor serial e porta fenecedora de energia. Um exemplo de resultado

pode ser visto na Figura 4.7.

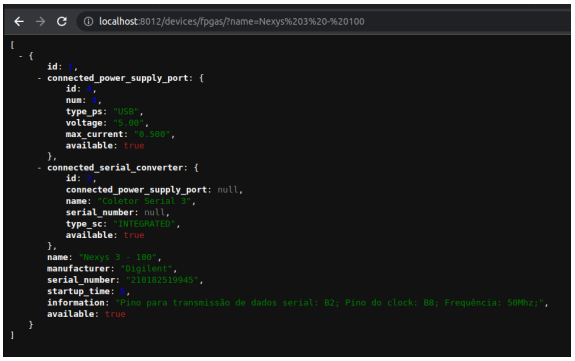
Figura 4.7: Resultado da API do Serviço *SPIDER-Device Manager*. Fonte: O autor.

Também é possível utilizar filtros na busca. Pode-se filtrar por ‘serial_number’ da placa (Figura 4.8a) ou por ‘name’ da placa (Figura 4.8b),

Figura 4.8: Resultado da API do Serviço *SPIDER-Device Manager* com filtros. Fonte: O autor.



(a) Filtro por ‘serial_number’.



(b) Filtro por ‘name’.

4.2 Cadastro de dispositivos no serviço SPIDER-Device Manager

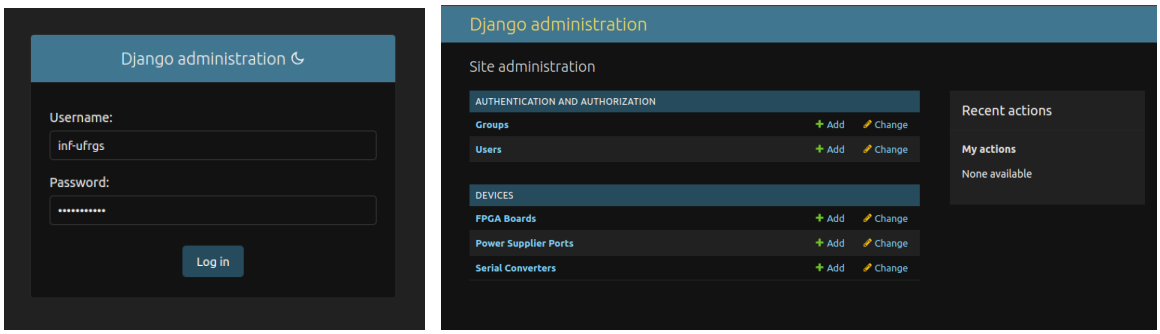
Nesta seção será mostrado o resultado de interação com a interface administrativa do Serviço *SPIDER-Device Manager*.

Para cadastrar dispositivos, basta acessar o *endpoint* ‘/admin’ e realizar o login

com usuário e senha (Figura 4.9a). Novos usuários podem ser cadastrados pelo administrador do sistema, possibilitando, assim, que existam mais de uma pessoa responsável por controlar o registro de dispositivos ofertados para uso remoto. Quando autenticado, o usuário terá acesso à página administrativa.

Como pode ser visto na Figura 4.9b existem algumas opções que podem ser selecionadas. Em *Users* é possível cadastrar novos usuários. Já em *Groups* é possível criar grupos personalizados de permissões e vincular um usuário a um ou mais grupos. Mais abaixo, temos as configurações de dispositivos. Acessando elas é possível cadastrar, editar ou excluir fontes fornecedoras de energia, placas com FPGA e conversores seriais.

Figura 4.9: Acessando a interface administrativa do Serviço *SPIDER-Device Manager*. Fonte: O autor.

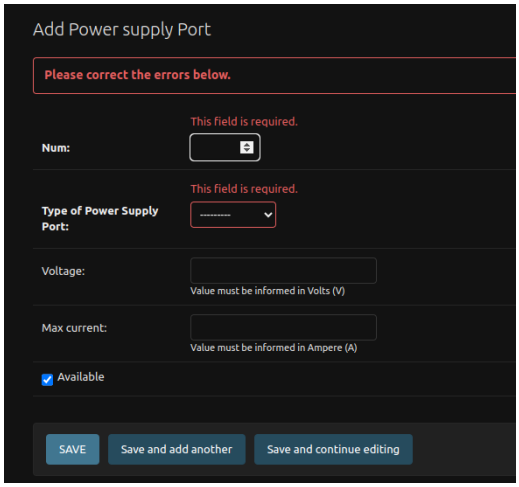


(a) Tela de login.

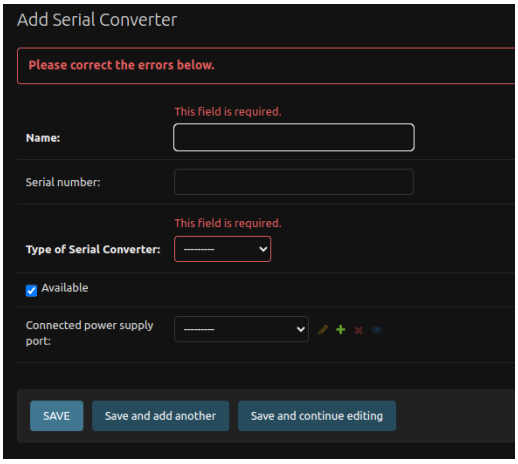
(b) Tela inicial.

No momento de cadastrar novos dispositivos, o sistema se mostra bastante robusto, fazendo validações para impedir que usuários deixem de inserir informações ou insiram dados incorretos. Essas validações funcionando na prática podem ser vistas na Figura 4.10.

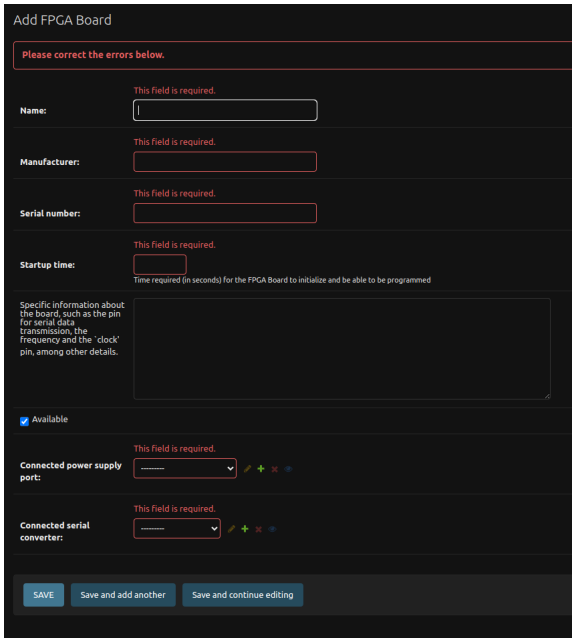
Figura 4.10: Validação de dados do *'SPIDER-Device Manager'*. Fonte: O autor.



(a) Validação dos dados de *'Power Supply Port'*.



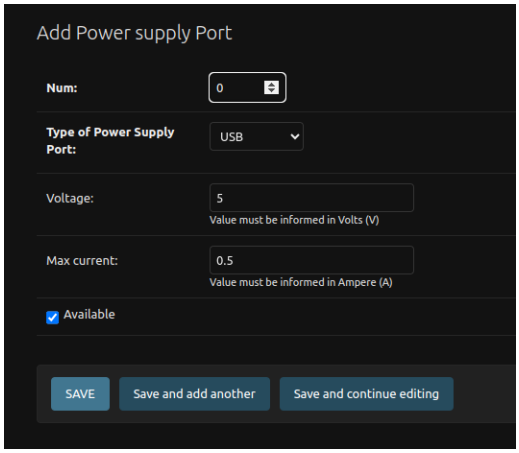
(b) Validação dos dados de *'Serial Converter'*



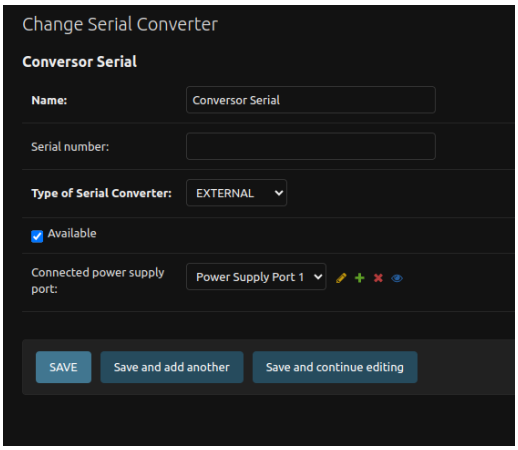
(c) Validação dos dados de *'FPGA Board'*

As imagens da Figura 4.11, mostram exemplos nos quais os dados são inseridos corretamente em cada campo. Pode-se notar que foi cadastrada uma fonte de fornecimento de energia do tipo USB, e ela foi atribuída a porta 0. Também foi cadastrado um conversor serial do tipo externo. Foi informado que ele está conectado a fonte de fornecimento de energia 1 (cadastrada previamente). Por fim, foi cadastrada uma placa com o nome de 'Basys 2 - 100'. Foi informado que essa placa está conectada a fonte de fornecimento de energia 0 e ao conversor serial criado na anteriormente.

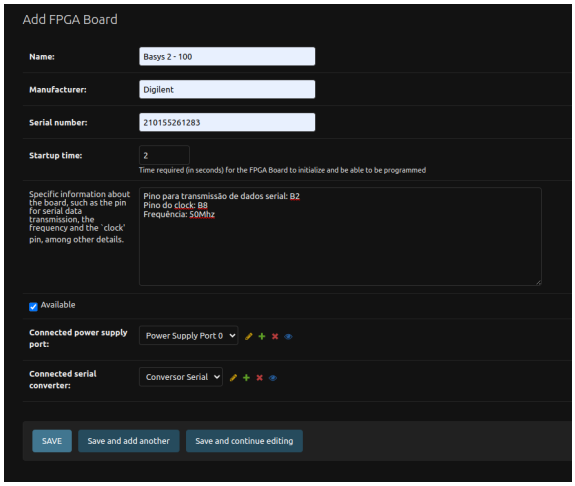
Figura 4.11: Cadastro de dados em *'SPIDER-Device Manager'*. Fonte: O autor.



(a) Cadastro dos dados em *'Power Supply Port'*.



(b) Cadastro dos dados em *"Serial Converter'*



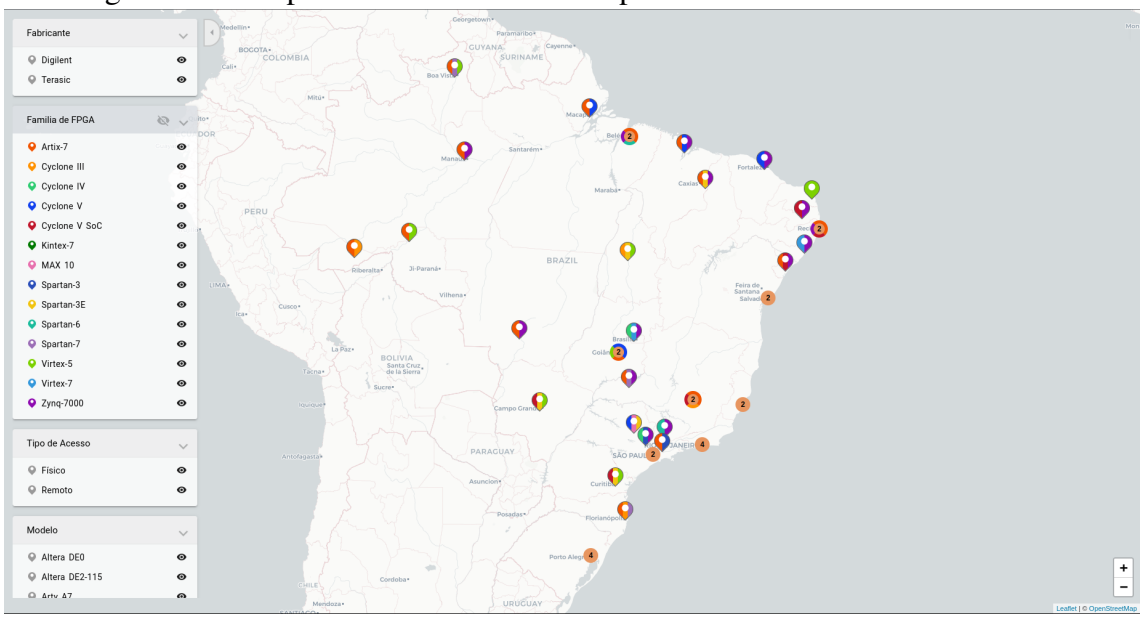
(c) Cadastro dos dados em *'FPGA Board'*

4.3 Programação de Placas de prototipagem com FPGA remotamente

Nesta seção será mostrado como é realizada a programação de uma placa remotamente utilizando o Mapa de Inventário juntamente com o serviço SPIDER.

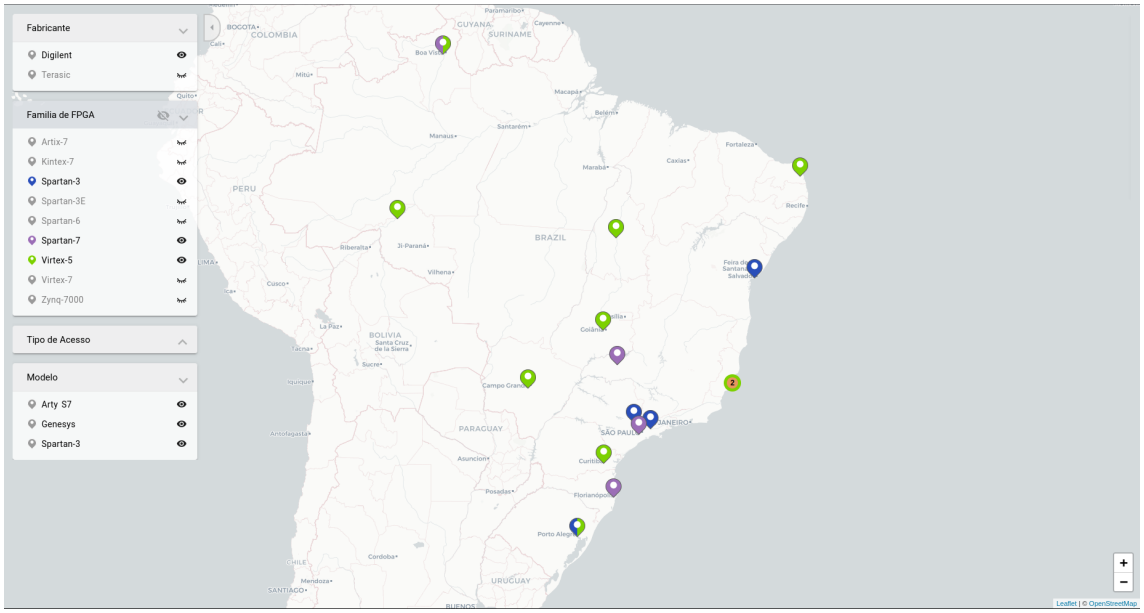
No momento que o usuário acessar o mapa, é possível ver uma tela semelhante a mostrada na Figura 4.12. Estes dados são fictícios, mostrados somente para ilustrar o funcionamento do sistema.

Figura 4.12: Mapa de Inventário exibindo placas cadastradas. Fonte: O autor.



Uma funcionalidade importante são os filtros. Com eles, é possível selecionar quais dispositivos são relevantes para serem exibidos no mapa, e com isso facilitar a busca por equipamentos. A Figura 4.13 mostra um exemplo de utilização de filtros, nos quais foram selecionados para exibir dispositivos da fabricante Digilent e FPGAs da família Spartan-3, Spartan-7 e Virtex-5.

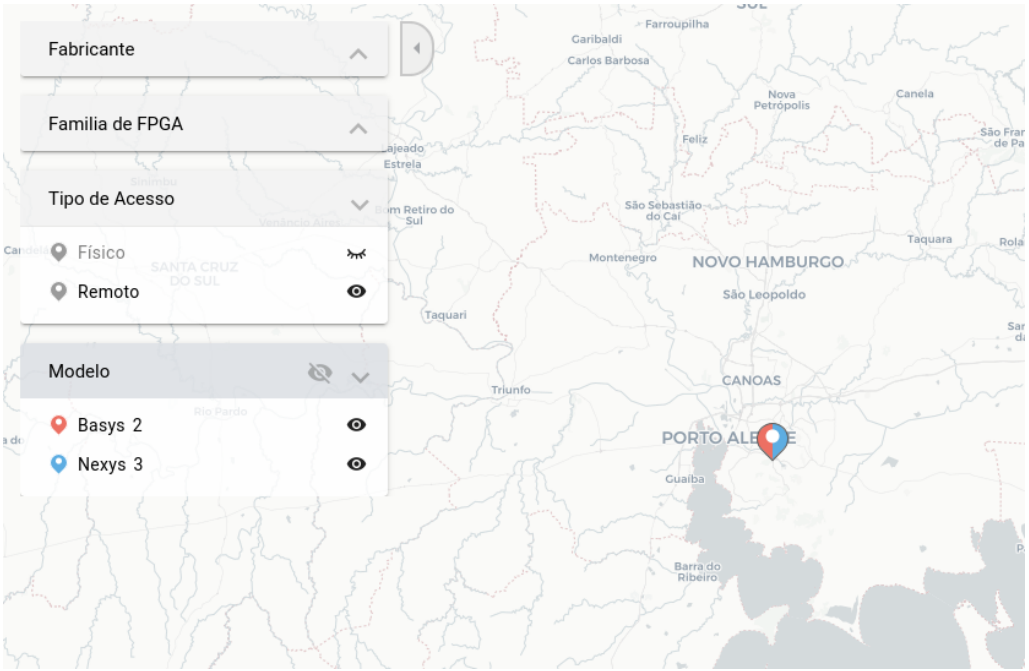
Figura 4.13: Utilização de filtros no Mapa de Inventário. Fonte: O autor.



Essa funcionalidade foi utilizada também para permitir que os utilizadores consigam encontrar de forma mais simples dispositivos que podem ser programados remotamente. O menu ‘Tipo de acesso’ permite que usuários filtrem por dispositivos disponíveis

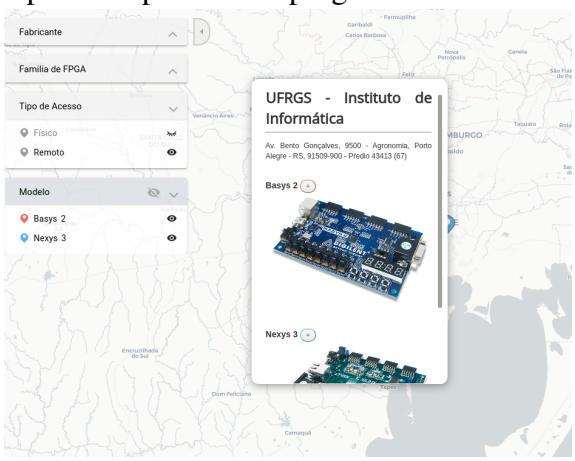
para uso físico ou remoto. Na figura 4.14 é utilizado um filtro para exibir somente dispositivos para uso remoto. Como pode ser visto, é mostrado somente o nodo do Instituto de Informática da UFRGS (INF-UFRGS), pois estes dados em específico são reais. Foram conectadas duas placas Basys 2 e uma placa Nexys 3 no sistema para que elas possam ser programadas remotamente.

Figura 4.14: Filtro para exibir placas com FPGA que podem ser programados remotamente. Fonte: O autor.



Ao clicar no nodo do INF-UFRGS é possível ver imagens das placas disponíveis e, caso ela possa ser programada remotamente, terá um ícone com o sinal de 'mais' (Figura 4.15).

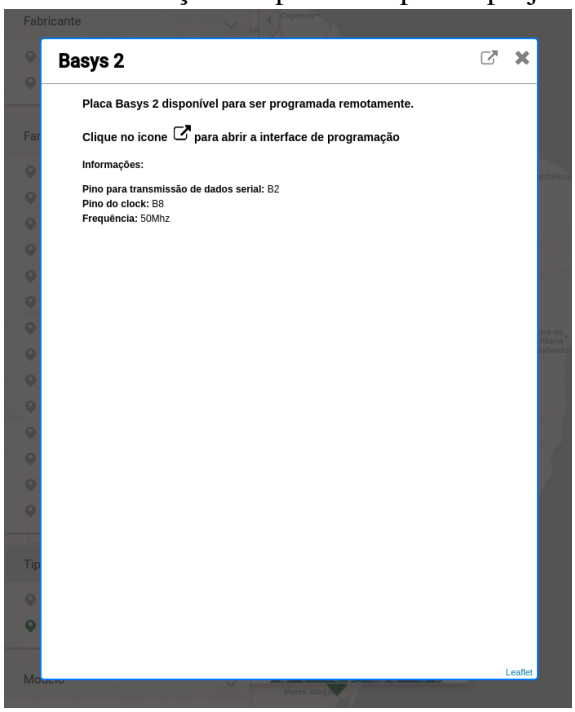
Figura 4.15: Placas disponíveis para serem programados remotamente. Fonte: O autor.



Ao clicar no ícone mencionado, é aberto um *popup* com informações importantes

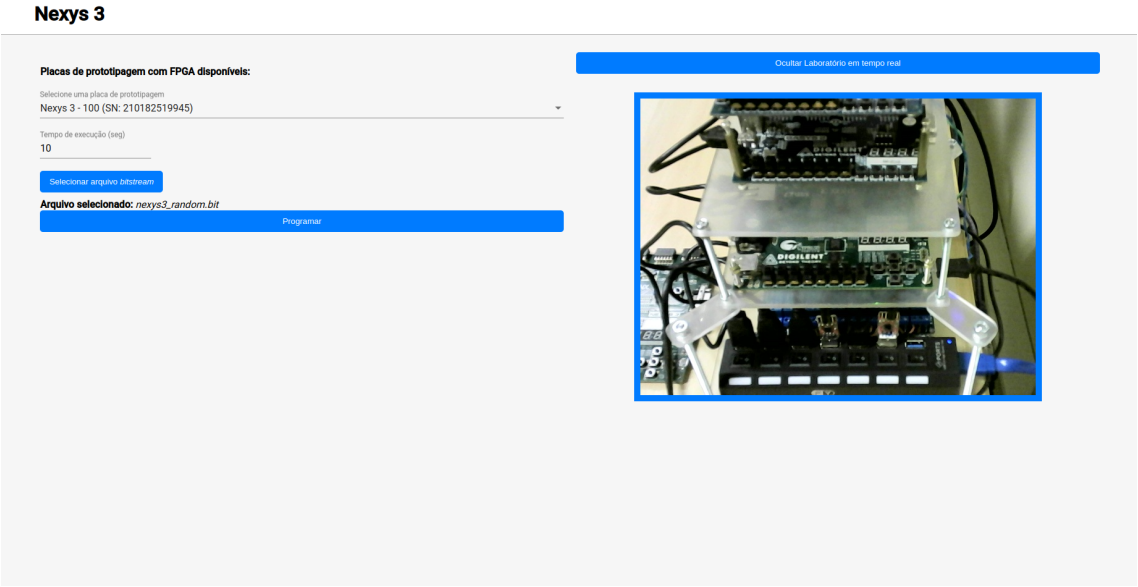
para a criação da descrição de hardware (Figura 4.16).

Figura 4.16: *Popup* com informações importantes para o projetista. Fonte: O autor.



Quando clicado no ícone de 'link externo' dentro do *popup*, a tela de programação é aberta, como mostrado na Figura 4.17

Figura 4.17: Interface para programação de placas com FPGA e coleta de dados. Fonte: O autor.

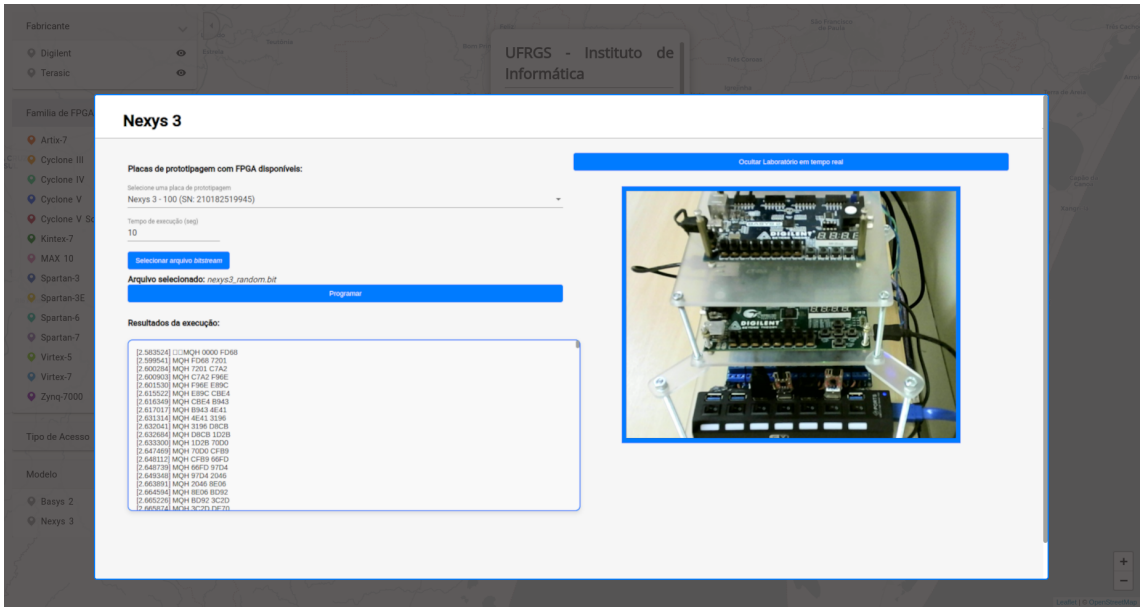


4.3.1 Execução e coleta de dados das placas Basys 2 e Nexys 3

Foram executados alguns códigos nas placas para certificar a possibilidade de coleta de códigos. Os testes foram realizados com sucessos e os dados foram coletados corretamente.

Na Figura 4.18 são mostrados os resultados de execução de uma descrição que implementa na FPGA Nexys 3 um gerador de números aleatórios a partir de uma 'semente'. Muitos dados foram coletados durante 10 segundos, porém como o espaço do campo de resultados é limitado, consegue-se apenas ver uma pequena parte na figura. Entretanto, a caixa de resultados possui um *scroll*, possibilitando que o usuário possa rolar a página, ver ou até mesmo salvar os dados.

Figura 4.18: Resultados do gerador de números aleatórios executado na placa Nexys 3. Fonte: O autor.



As Figuras 4.19 e 4.20, mostram os resultados de um contador que incrementa o valor 1 a cada um segundo. Esse contador foi também executado nas placas Nexys 3 e Basys 2.

Figura 4.19: Resultados do contador executado na placa Nexys 3. Fonte: O autor.

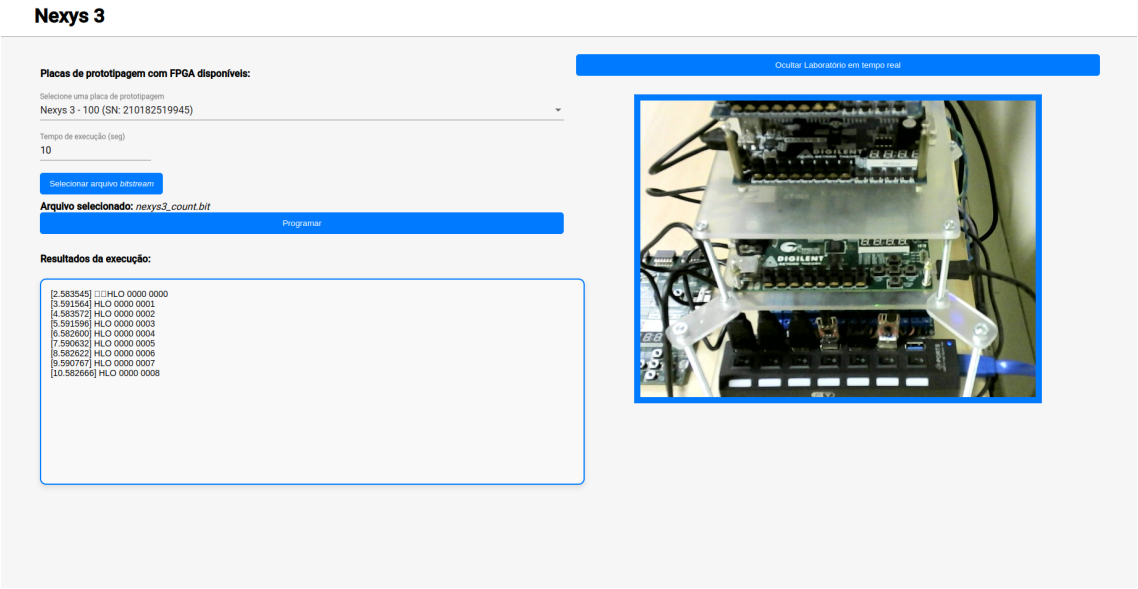
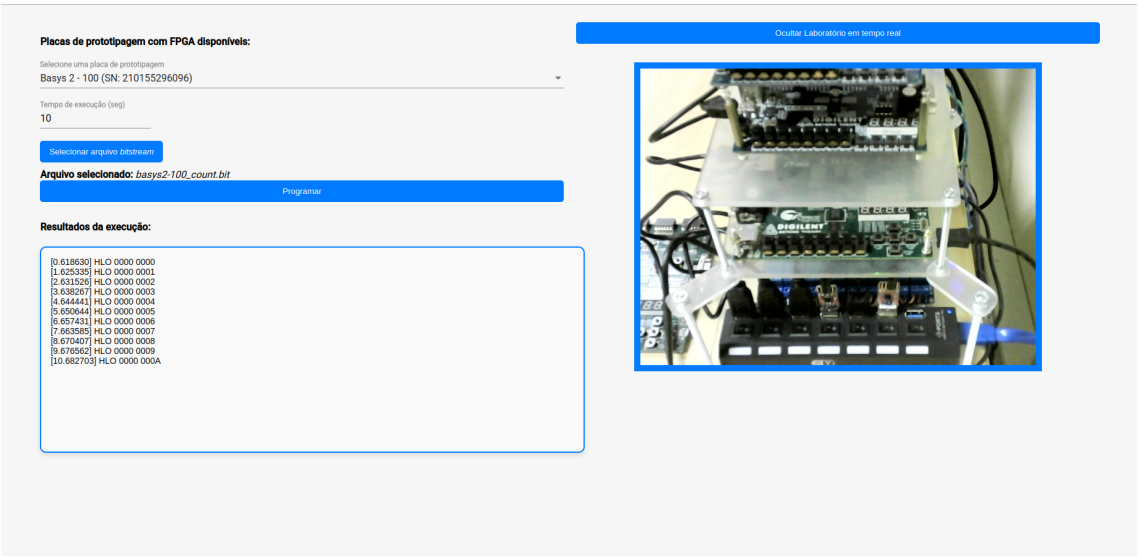


Figura 4.20: Resultados do contador executado na placa Basys 2. Fonte: O autor.



5 CONSIDERAÇÕES FINAIS

O presente trabalho possibilitou comprovar a possibilidade de realizar, em placas de prototipação com FPGA, a programação e coleta de dados de forma completamente remota. A infraestrutura criada mostrou-se robusta e com uma ótima manutenibilidade, já que durante o desenvolvimento deste trabalho foram realizadas diversas melhorias e o sistema mostrou-se apto à evolução. A arquitetura utilizando camadas favorece uma grande possibilidade de integração com outros sistemas. Por exemplo, o sistema poderia ser conectado a um novo *front-end* ou a um novo módulo de gerenciamento de energia sem muito esforço.

A infraestrutura utilizando *containers* permite que o sistema seja facilmente instanciado em laboratórios de outras instituições, bastando para isso que o servidor tenha um IP público para permitir conexões externas. Outra opção, seria realizar parcerias, por exemplo, com a Rede Nacional de Pesquisa (RNP), para utilizar redes de experimentação, facilitando o processo de conectar novos laboratórios à rede.

É importante salientar que, apesar dos resultados positivos, o sistema ainda não é compatível com todos os modelos de placas, já que geralmente cada fabricante possui o seu próprio software para programar a placa. Entretanto, como mencionado, o sistema é altamente flexível e adicionar suporte a novas placas é completamente viável.

5.1 Limitações

Assim como todo sistema criado a partir do zero, o SPIDER tem suas limitações. Mesmo que ele tenha sido planejado e as tecnologias permitam acesso simultâneo de vários usuários, o sistema pode sofrer algumas instabilidades se dois usuários solicitarem, ao mesmo tempo, programar uma mesma placa. Isso se deve a limitações de hardware, pois no momento que uma placa é programada, ela não deve ser programada por outro usuário. Esses detalhes não foram aprofundados nesse trabalho, pois estava fora do escopo. Entretanto, seria importante ter um novo microsserviço responsável por alocar e bloquear o uso para outras pessoas de uma placa que foi escolhida para ser programada.

Outra limitação é em relação ao sistema de gerenciamento de energia. Nesse trabalho, foi utilizado um Arduino Uno como controlador, mas ele possui limitação de portas digitais e analógicas. Caso um laboratório possua muitas placas, uma opção seria utilizar um Arduino Mega, que possui uma quantidade maior de portas digitais e análgicas. Em

situações mais específicas, poderia ser utilizado até outro tipo de microcontrolador.

5.2 Trabalhos Futuros

Como mencionado acima, um trabalho futuro muito valioso seria desenvolver um serviço capaz de lidar com acessos e tentativas de programação simultâneas. No momento o SPIDER não possui uma grande quantia de usuários, mas em um futuro, caso venha a ter, um sistema desse tipo seria essencial.

Outro trabalho futuro, muito valioso e importante, seria implementar um sistema de controle de acesso e permissões. Que seja possível cadastrar usuários, quais laboratórios e placas eles têm permissão para utilizar.

Como o sistema foi projetado para ser escalável, outro trabalho interessante seria implantar o sistema em novas instituições, permitindo que usuários interessados tenham acesso a mais modelos de placas com FPGA.

Visando uma maior facilidade ao administrador do laboratório, poderia existir um módulo capaz de identificar automaticamente os dispositivos conectados no sistema e cadastrá-los automaticamente no serviço SPIDER-Device Manager (3.3.9).

Atualmente existem dois bancos de dados, um no Mapa de Inventário (2.3) e outro no serviço SPIDER-Device Manager (3.3.9). Outra funcionalidade seria existir um módulo de sincronização entre os dois bancos. Ou seja, alterações feitas no SPIDER-Device Manager serem propagadas automaticamente para o Mapa de Inventário.

REFERÊNCIAS

ABANTO, D. et al. Implementation of a web-based interface for remote access to a fpga board. In: **2022 IEEE Engineering International Research Conference (EIRCON)**. [S.l.: s.n.], 2022. p. 1–4.

Amazon. **Instâncias F1 do Amazon EC2**. 2023. Disponível na internet em: <https://aws.amazon.com/pt/ec2/instance-types/f1/>. Acesso em: 13 de agosto de 2023.

Angular Team. **What is Angular**. 2023. Disponível na internet em: <https://angular.io/guide/what-is-angular>. Acesso em: 2 de Agosto de 2023.

Arduino. **Arduino UNO R3**. 2023. Disponível na internet em: <https://docs.arduino.cc/hardware/uno-rev3>. Acesso em: 10 de agosto de 2023.

ARDUINO. **Introduction to Arduino - Official Guide**. 2023. Disponível na internet em: <https://www.arduino.cc/en/Guide/Introduction>. Acessado em: 5 de agosto de 2023.

AUTHORITY, P. P. **Python Package Index (PyPI)**. 2023. Disponível na internet em: <https://pypi.org/>. Acessado em: 5 de agosto de 2023.

Basic Electronics Tutorials. **Electrical Relay and Solid State Relay**. 2022. Disponível na internet em: <https://www.electronics-tutorials.ws/io/io_5.html>. Acesso em: 6 de agosto de 2023.

BETZ, V.; ROSE, J.; MARQUARDT, A. **Architecture and CAD for deep-submicron FPGAs**. [S.l.]: Springer Science & Business Media, 2012.

BLOCHWITZ, C. et al. Remedula - remote education laboratory for fpga design technology. In: **2022 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2022. p. 1773–1777.

Digilent. **Nexys 3 Reference Manual**. 2021. Disponível na internet em: <https://digilent.com/reference/programmable-logic/nexys-3/reference-manual>. Acesso em: 6 de agosto de 2023.

Digilent. **ZedBoard Zynq-7000 Development Board Reference Manual**. 2021. Disponível na internet em: <https://digilent.com/reference/programmable-logic/zedboard/reference-manual>. Acesso em: 6 de agosto de 2023.

Django Software Foundation. **Django FAQ: General**. 2023. Disponível na internet em: <https://docs.djangoproject.com/en/4.2/faq/general/>. Acesso em: 2 de Agosto de 2023.

Django Software Foundation. **Django Project Documentation: Security Topics**. 2023. <https://docs.djangoproject.com/en/4.2/topics/security/>.

Docker Inc. **Docker Compose overview**. 2023. Disponível na internet em: <https://docs.docker.com/compose/>. Acesso em: 8 de agosto de 2023.

DRAGONI, N. et al. Microservices: yesterday, today, and tomorrow. **Present and ulterior software engineering**, Springer, p. 195–216, 2017.

DUMPLETON, G.; EBY, P. J. **PEP 3333 – Python Web Server Gateway Interface v1.0.1**. 2023. Disponível na internet em: <https://peps.python.org/pep-3333/>. Acessado em: 5 de agosto de 2023.

FOUNDATION, P. S. **General Python FAQ**. 2023. Disponível na internet em: <https://docs.python.org/3/faq/general.html#what-is-python>. Acessado em: 5 de agosto de 2023.

GOMES, L. et al. Remote experimentation for introductory digital logic course. In: **2009 3rd IEEE International Conference on E-Learning in Industrial Electronics (ICELIE)**. [S.l.: s.n.], 2009. p. 98–103.

Google. **Material Design for Angular**. 2023. Disponível na internet em: <https://material.angular.io/>. Acesso em: 2 de Agosto de 2023.

HAUCK, S.; DEHON, A. **Reconfigurable computing: the theory and practice of FPGA-based computation**. [S.l.]: Elsevier, 2010.

HOLOVATY, A.; KAPLAN-MOSS, J. **The definitive guide to Django: Web development done right**. [S.l.]: Apress, 2009.

KARYONO; WICAKSANA, A. Teaching microprocessor and microcontroller fundamental using fpga. In: **2013 Conference on New Media Studies (CoNMedia)**. [S.l.: s.n.], 2013. p. 1–5.

KUON, I.; ROSE, J. Measuring the gap between fpgas and asics. In: **Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays**. [S.l.: s.n.], 2006. p. 21–30.

LABS, S. **CP2102 USB-to-UART Bridge - Specifications**. 2023. Disponível na internet em: <https://www.silabs.com/interface/usb-bridges/classic/device.cp2102?tab=specs>. Acessado em: 21 de agosto de 2023.

MAYOZ, C. A. et al. Fpga remote laboratory: experience of a shared laboratory between upna and unifesp. In: **2020 XIV Technologies Applied to Electronics Teaching Conference (TAEE)**. [S.l.: s.n.], 2020. p. 1–8.

MOMJIAN, B. **PostgreSQL: introduction and concepts**. [S.l.]: Addison-Wesley New York, 2001.

NEDELCU, C. **Nginx http server**. [S.l.]: Packt Publishing, 2013.

NEWMAN, S. **Building microservices**. [S.l.]: "O'Reilly Media, Inc.", 2021.

NICKOLOFF, J.; KUENZLI, S. **Docker in action**. [S.l.]: Simon and Schuster, 2019.

PALLETS. **Foreword - Flask Documentation (2.0.x)**. 2023. Disponível na internet em: <https://flask.palletsprojects.com/en/2.0.x/foreword/>. Acessado em: 5 de agosto de 2023.

PALLETS. **A Minimal Application - Flask Documentation (2.0.x)**. 2023. Disponível na internet em: <https://flask.palletsprojects.com/en/2.0.x/quickstart/#a-minimal-application>. Acessado em: 5 de agosto de 2023.

PALLETS. **Routing - Flask Documentation (2.0.x)**. 2023. Disponível na internet em: <https://flask.palletsprojects.com/en/2.0.x/quickstart/#routing>. Acessado em: 5 de agosto de 2023.

PETERS, T. **PEP 20 – The Zen of Python**. 2023. Disponível na internet em: <https://peps.python.org/pep-0020/>. Acessado em: 5 de agosto de 2023.

PostgreSQL Global Development Group. **PostgreSQL**. 2023. Disponível na internet em: <https://www.postgresql.org/>. Acesso em: 1 de agosto de 2023.

POSTMAN, I. **Postman API Platform**. 2023. Disponível na internet em: <https://www.postman.com/>. Acessado em: 21 de agosto de 2023.

QASSEM, L. M. A. et al. A remote fpga laboratory as a cloud microservice. In: **2020 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2020. p. 1–5.

RAVINDRAN, A. **Django Design Patterns and Best Practices**. [S.l.]: Packt Publishing Ltd, 2015.

SAABITH, A. S.; VINOTHRAJ, T.; FAREEZ, M. Popular python libraries and their application domains. **International Journal of Advance Engineering and Research Development**, v. 7, n. 11, 2020.

Silicon Labs. **CP2102 Classic USB to UART Bridge**. 2023. Disponível na internet em: <https://www.silabs.com/interface/usb-bridges/classic/device.cp2102?tab=specs>. Acesso em: 13 de agosto de 2023.

SOARES, J.; LOBO, J.; DEEC, F. A remote fpga laboratory for digital design students. In: **7th portuguese meeting on reconfigurable systems (REC 2011)**. [S.l.: s.n.], 2011. p. 95–98.

Tom Christie. **Django REST framework**. 2023. Disponível na internet em: <https://www.django-rest-framework.org/>. Acesso em: 1 de agosto de 2023.

TRIMBERGER, S. M. **Field-programmable gate array technology**. [S.l.]: Springer Science & Business Media, 2012.

ULUCA, D. **Angular 6 for Enterprise-Ready Web Applications: Deliver production-ready and cloud-scale Angular web apps**. [S.l.]: Packt Publishing Ltd, 2018.

UNICAMP, E. **Arduino: Entenda mais sobre essa versátil plataforma**. 2023. Disponível na internet em: <https://3eunicamp.com/arduino-entenda-mais-sobre-essa-versatil-plataforma/>. Acessado em: 5 de agosto de 2023.

VINCENT, W. S. **Django for Beginners: Build websites with Python and Django**. [S.l.]: WelcomeToCode, 2021.