UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Maria Victória Mundim Saramago

Desenvolvimento de uma aplicação Web para acesso remoto a computadores de bordo utilizados no ramo da automação agrícola

Florianópolis
2022

Maria Victória Mundim Saramago

Desenvolvimento de uma aplicação Web para acesso remoto a computadores de bordo utilizados no ramo da automação agrícola

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Ronaldo dos Santos Mello, Dr.
Co-orientador: Luiz Henrique Zambom Santana, Dr.
Supervisor: Hugo Pereira Fagundes, Eng.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Saramago, Maria Victória Mundim
Desenvolvimento de uma aplicação Web para acesso remoto a computadores de bordo utilizados no ramo da automação agrícola / Maria Victória Mundim Saramago ; orientador, Ronaldo dos Santos Mello, coorientador, Luiz Henrique Zambom Santana, 2022.
80 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2022.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Automação agrícola. 3. Ferramentas de suporte. 4. Sistemas embarcados. I. Mello, Ronaldo dos Santos. II. Santana, Luiz Henrique Zambom. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. IV. Título.

Maria Victória Mundim Saramago

Desenvolvimento de uma aplicação Web para acesso remoto a computadores de bordo utilizados no ramo da automação agrícola

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 13 de dezembro de 2022.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Ronaldo dos Santos Mello, Dr.
Orientador
UFSC/CTC/INE

Luiz Henrique Zambom Santana, Dr.
Co-orientador
UFSC/CTC/INE

Hugo Pereira Fagundes, Eng.
Supervisor
Hexagon

Eduardo Rauh Müller, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

Dedicado ao meu companheiro de quase uma década,
Eduardo Jorge Siridakis.

Resumo

Hexagon é uma empresa do ramo de tecnologia cujas soluções envolvem a aquisição de dados por meio de sensores e sistemas de posicionamento geo-espacial. A divisão de agricultura da Hexagon oferece soluções de controle e monitoramento para operação de máquinas agrícolas, tendo como produto central um computador de bordo, ou display, que é instalado nas máquinas e se comunica com a Internet para transmitir dados de telemetria. A equipe de suporte da empresa realiza sessões de acesso remoto para a investigação de eventuais problemas em campo, utilizando duas ferramentas, sendo uma delas uma página web que apresenta a tela dos displays em tempo real, e outra um script elaborado para permitir o acesso e controle do terminal dos displays remotamente, para a coleta de *logs* e envio de comandos que podem ser úteis durante a investigação dos problemas. Por conta de limitações encontradas na segunda ferramenta, e o interesse em integrar os sistemas de suporte da empresa, propôs-se o desenvolvimento de uma nova solução para acesso remoto via terminal, que seja integrada à plataforma de *streaming* da tela, com controle de acesso e armazenamento de logs de sessão. Estudos de mercado foram realizados para determinar quais tecnologias são empregadas em soluções de acesso remoto via terminal para sistemas embarcados, e uma estrutura preliminar foi definida, composta de um agente de software embarcado e um componente de aplicação web, que se comunicam utilizando o protocolo MQTT e a tecnologia *WebSocket*. Um protótipo foi implementado e integrado à aplicação web já existente, e dados de performance, como latência da conexão, tráfego de dados e alocação de recursos do display, foram coletados e comparados com uma solução comercial já estabelecida para o mesmo propósito. O projeto atingiu os objetivos propostos, e algumas melhorias e implementações futuras foram identificadas na etapa de testes.

Palavras-chave: Automação agrícola. Ferramentas de suporte. Sistemas embarcados.

Abstract

Hexagon is a technology company, developing solutions on the field of data acquisition through sensors and geospatial systems. Hexagon's agriculture division offers control and monitoring solutions for the operation of agricultural machines, and its main product is an onboard computer, or display, which is installed in the machines and accesses the Internet to transmit telemetry data. The company's support team performs remote access sessions to investigate possible problems in the field, using two tools, one of which is a web page that shows the screen of the displays in real time, and the other being a script designed to allow access and terminal control of displays remotely, for collecting *logs* and sending commands that may be useful during problem investigation. Due to limitations found in the second tool, and the interest in integrating all of the company's support systems, a new solution for remote access via terminal was proposed, which is integrated to the current remote access platform, with access control and session log storage. Market studies were carried out to determine which technologies are employed in remote access solutions via terminal for embedded systems, and a preliminary structure was defined, composed of an embedded software agent and a web application component, which communicate using the MQTT protocol. and the *WebSocket* technology. A prototype was implemented and integrated into the existing web application, and performance data, such as connection latency, data traffic and display resource allocation, were collected and compared with an established commercial solution for the same purpose. The project was successful in achieving the outlined objectives, and some improvements and future implementations were identified in the testing phase.

Keywords: Agricultural automation. Support Tools. Embedded systems.

Lista de figuras

Lista de tabelas

Sumário

1 Introdução

Este capítulo contém uma breve contextualização do projeto desenvolvido, apresentando o problema a ser solucionado, os objetivos gerais e específicos do projeto, e a metodologia selecionada para o desenvolvimento. Ao final, apresenta-se a organização do documento.

1.1 Descrição do problema

A automação agrícola, um dos conceitos base da tendência conhecida como agricultura 4.0, consiste no emprego de tecnologias auxiliares durante as operações no campo, visando o controle das máquinas (tratores, semeadoras ou colhedoras, por exemplo), o mapeamento do solo e das condições de plantio, e o registro das atividades produtivas e improdutivas. Como resultado, os produtores podem regular o padrão de qualidade da operação, economizar nos gastos em insumos como fertilizantes e defensivos e melhorar significativamente sua produtividade e desempenho.

A divisão de agricultura do grupo Hexagon[1] oferece soluções de automação agrícola para clientes corporativos e parceiros de revenda nacionais e internacionais, tendo como produto principal um computador de bordo com três modelos disponíveis Ti5, Ti7 e Ti10, denominados displays. O display é o componente central das soluções da divisão de agricultura, disponibilizando uma interface para configuração, controle e monitoramento das máquinas agrícolas e dos equipamentos que integram a operação no campo.

Eventuais pedidos de suporte relacionados aos displays que estão em campo podem ser tratados pela assistência técnica, tratando-se de questões relacionadas a hardware; já problemas de configuração ou *bugs* são analisados pelas equipes de suporte, serviços e engenharia de aplicação.

Dependendo do tipo de suporte prestado, é necessário acessar remotamente o display que apresenta problemas para realizar configurações específicas de depuração, visando a coleta de informações para a investigação, além da realização de testes, que servem de respaldo à resolução do problema pelos times de desenvolvimento.

Uma solução de acesso remoto foi lançada pela empresa no começo de 2022, chamada de AgrOn Acesso Remoto. No entanto, essa solução atualmente permite apenas o *streaming* da tela do display. Outras *features*, como a possibilidade de controlar a interface display remotamente, estão em processo de validação. E, especificamente, a coleta de alguns *logs* de depuração não é possível pela interface, apenas pelo terminal. Por isso, o time de entrega de soluções e suporte elaborou um script para realizar o acesso remoto aos displays em modo de terminal, utilizando SSH (*Secure Shell*). Essa ferramenta interna possui limitações, como a ausência de controle de acesso e

[1] https://hexagon.com/

uma taxa de falhas de conexão consideravelmente alta, relacionada a senhas de SSH incorretas, o que inviabiliza o estabelecimento da conexão remota. Essas limitações serão discutidas com mais detalhes ao longo do documento.

Buscando o fortalecimento dos processos de suporte da empresa, deseja-se criar um sistema centralizado para o acesso ao terminal dos displays, que esteja integrado à ferramenta de acesso remoto já disponível. Esse sistema será acessado por colaboradores da Hexagon e auxiliará nas tratativas de suporte e na validação de novas soluções em campo. É desejável que esse sistema possua controle de acesso e armazene o histórico de sessões e comandos enviados aos displays. Visando a execução deste projeto, um objetivo geral e alguns objetivos específicos foram propostos, descritos abaixo.

1.2 Objetivos

1.2.1 Objetivo geral

Integrar o acesso ao terminal dos displays da Hexagon à aplicação web de acesso remoto já existente.

1.2.2 Objetivos específicos

- Eliminar a dependência da senha de SSH para o acesso ao terminal.
- Implementar um controle de acesso à nova funcionalidade.
- Coletar logs de data/hora de acesso e registrar os comandos enviados durante a sessão.
- Obter estatísticas de performance da aplicação.

1.3 Metodologia

A metodologia adotada para o desenvolvimento do projeto é baseada no *framework* UP (*Unified Process*, processo unificado). As etapas de projeto no UP são: concepção, elaboração, construção e transição (WAZLAWICK, 2016). Este documento deve cobrir as etapas de concepção até o início da transição. Resumidamente, as atividades que se encaixam em cada etapa são descritas a seguir.

1.3.1 Concepção

Na etapa de concepção, serão definidos o escopo de projeto, o nível de modularidade e os requisitos (funcionais e não-funcionais), que serão levantados por meio da descrição de casos de uso de alto nível. Para embasar essa etapa, as seguintes atividades devem ser cumpridas previamente:

- Estudo da ferramenta existente por meio da documentação disponível na base de conhecimento da empresa. A execução do projeto está fortemente associada com o estudo da documentação e consulta aos desenvolvedores do AgrOn Acesso Remoto, já que os resultados nele obtidos deverão integrar um sistema que já está em operação;
- Pesquisa de mercado para obter informações a respeito das práticas mais comuns associadas ao gerenciamento de acesso remoto a sistemas embarcados. Serão estudadas alternativas comerciais e uma delas será adotada como linha de base para avaliar os resultados.

1.3.2 Elaboração

Na etapa de elaboração, os casos de uso obtidos anteriormente serão refinados, estabelecendo uma sequência de passos que define cada um deles. Para ilustrar esses passos, serão elaborados diagramas auxiliares baseados em UML *Unified Modeling Language*, como diagramas de máquina de estados e de sequência.

1.3.3 Construção

Durante a etapa de construção, o sistema será efetivamente implementado, e questões associadas à implementação serão abordadas, como tecnologias, serviços e *frameworks* utilizados. Diagramas de classes, arquitetura e *deployment* serão empregados para ilustrar a interação entre os módulos projetados.

1.3.4 Transição

A etapa de transição marca o final do desenvolvimento e a realização de testes, a partir dos quais serão coletados dados a respeito da performance da aplicação, tanto em termos de recursos computacionais do display, como memória e uso de CPU no sistema embarcado, cuja aplicação principal consome recursos computacionais consideráveis. Também será avaliada a performance de rede, utilizando quesitos como o tempo de resposta aos comandos e consumo médio de dados, tendo em mente que alguns clientes fazem uso de redes móveis com alta latência.

Os atributos de performance avaliados deverão garantir que o sistema de acesso remoto não adiciona *overhead* significativo aos processos já executados no sistema embarcado, e é capaz de estabelecer (e manter) conexão com displays nos cenários de rede encontrados durante a operação em campo.

1.4 Estrutura do documento

No capítulo 2, serão apresentadas informações gerais sobre a empresa Hexagon, com foco na divisão de agricultura e nas as atividades desempenhadas pela equipe de suporte e entrega de soluções. Também será apresentada uma visão geral da linha de displays Ti5, Ti7 e Ti10, direcionada às especificações de hardware, ao sistema operacional e aos serviços de software embarcado.

O capítulo 3 apresentará as ferramentas de acesso remoto utilizadas pela equipe de suporte da Hexagon: o AgrOn Acesso Remoto e o *ReverseSSH*, script elaborado pela equipe de suporte para acessar os displays, e quais são as suas limitações. Também serão levantadas as as estratégias utizadas no mercado atualmente para gerenciar o acesso remoto a dispositivos, com foco em uma ferramenta específica, que será utilizada como linha de base para a execução do projeto.

O capítulo 4 incluirá detalhes sobre a modelagem do projeto, cobrindo a etapa de Elaboração do UP. Serão apresentados os requisitos funcionais e não-funcionais, modelos conceituais, definições gerais de arquitetura e diagramas UML.

O capítulo 5 tratará do desenvolvimento e detalhes relacionados à implementação do projeto, compondo a etapa de Construção do UP. Os serviços da AWS relevantes para a execução do projeto serão listados e suas principais funções e integrações serão descritas. Também serão apresentadas algumas informações sobre o *framework Angular* para desenvolvimento web, e sobre controle de entrada/saída em processos de terminal.

No capítulo 6, serão apresentados os resultados da etapa de Transição, que engloba a validação dos requisitos funcionais e não funcionais por meio de testes com o projeto, incluindo a coleta de dados a respeito da performance dos módulos do sistema.

Finalmente, no capítulo 7, serão apresentadas as conclusões relacionadas ao desenvolvimento do projeto e possíveis desenvolvimentos futuros previstos e/ou identificados durante os testes.

2 Sobre a Hexagon e seus produtos

Este capítulo trata da empresa Hexagon, na qual o projeto foi realizado, e de seu principal produto, a linha de displays Ti5, Ti7 e Ti10.

A seção 2.1 apresenta a Hexagon corporativa e foca na divisão de agricultura, seu modelo de negócio, principais clientes e áreas de atuação, oferecendo uma breve descrição de seus principais produtos, e da função da equipe de suporte e entrega de soluções (*Solution Deployment*), aqui referida simplesmente como equipe de suporte.

Na seção 2.2, serão apresentados detalhes adicionais a respeito dos displays da Hexagon, como especificações de hardware e sistema operacional, e o processo de provisionamento de atualizações de software e de configuração.

2.1 Hexagon AB, o grupo Hexagon

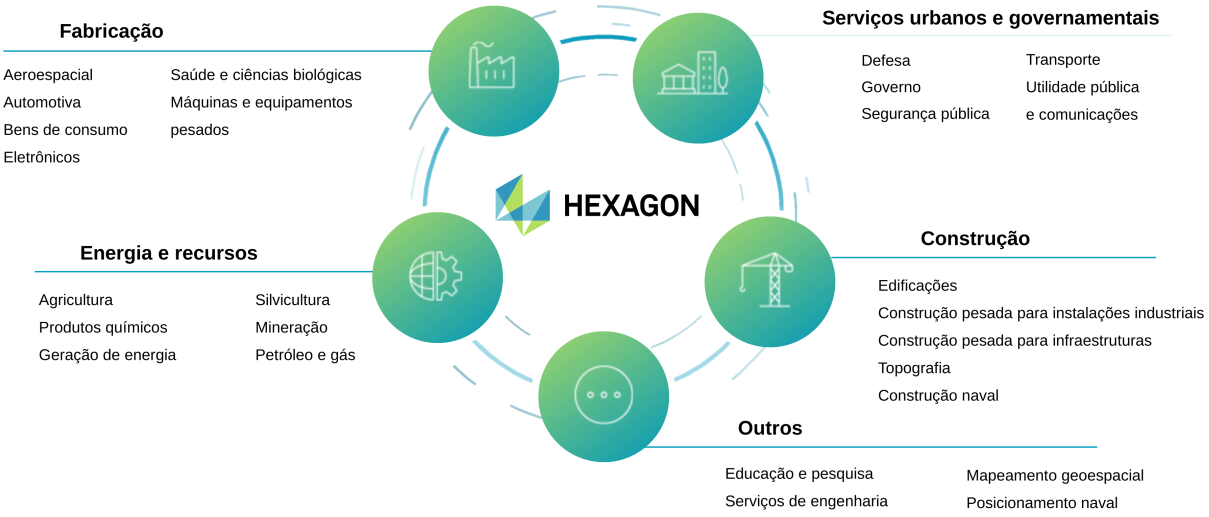
Hexagon AB é um grupo do ramo de tecnologia sediado em Estocolmo, Suécia, e fundado em 1992, que atualmente está presente em 50 países, totalizando cerca de 23.000 funcionários e apresentando faturamento líquido de aproximadamente 4,3 bilhões de euros (HEXAGON, c2022a). O grupo Hexagon é composto por unidades de negócio denominadas divisões, sendo cada divisão voltada a uma área de atuação específica. Destacam-se as áreas de geomensura, mineração, agricultura, segurança e infraestrutura. As soluções oferecidas pelas divisões da Hexagon compartilham um tema central: a aquisição de dados por meio de sensores e sistemas de posicionamento geo-espacial, como GNSS (*Global Navigation Satellite System*), e o processamento dos dados obtidos via software. A Figura 1 apresenta um diagrama com as principais áreas de atuação do grupo.

2.1.1 A divisão de agricultura da Hexagon

A divisão de agricultura da Hexagon, também conhecida como *Hexagon Agriculture*, possui sede localizada em Florianópolis, Santa Catarina. Existem escritórios adicionais em Ribeirão Preto (São Paulo) e Madrid (Espanha). Surgiu por meio da aquisição de três empresas em 2014: Arvus Tecnologia, ILab Sistemas e Leica Geosystems (HEXAGON, c2022b).

A divisão atende clientes corporativos nacionais e internacionais, como usinas de cana-de-açúcar e fábricas de papel e celulose, e também trabalha com os mercados *aftermarket* (pós-venda), no qual parceiros oferecem os produtos Hexagon para compra, e OEM *(Original Equipment Manufacturer)*, no qual fabricantes de equipamentos agrícolas integram os produtos da Hexagon às suas máquinas. Em geral, a divisão desenvolve soluções que auxiliam na gestão, planejamento, monitoramento e otimização dos processos de cultivo, colheita e transporte de matéria-prima.

Figura 1 – Áreas de atuação da Hexagon.



Fonte: Adaptado de documentação interna da empresa.

No restante deste documento, a Hexagon Agriculture será referida simplesmente como “a Hexagon” ou “a empresa”. Quando for necessário fazer referência à Hexagon de forma geral, o termo “grupo Hexagon” será utilizado.

2.1.2 Produtos

O principal produto da Hexagon é a linha de displays Ti5, Ti7 e Ti10, assim denominados por possuírem telas de 5, 7 e 10 polegadas, respectivamente[1]. As soluções oferecidas pela Hexagon são amplamente baseadas nos dados coletados pelos displays. A Figura 2 apresenta o display Ti7 instalado na cabine de um trator agrícola.

O display é responsável por realizar a interface entre o operador da máquina e os diversos componentes (sensores, atuadores) instalados na mesma. Acessando os menus, é possível monitorar e registrar informações sobre o *status* da operação, além de realizar o controle das atividades de plantio, pulverização ou fertilização, por exemplo. Esse controle é realizado em conjunto com uma ECU (*Electronic Control Unit*), que pode ser da Hexagon ou de terceiros, de forma que a ECU realiza o controle a baixo nível (geração de sinal, estágio de potência, aquisição) e o display realiza o controle a alto nível (ganhos do controlador, *offset*, entre outros parâmetros de operação, dependentes do tipo de equipamento).

Além dos displays, também são produzidos componentes auxiliares para máquinas agrícolas, como sistemas de piloto automático e ECUs que permitem a integração do display com diversos implementos agrícolas. Implemento é a designação de um

[1] Mais detalhes em https://hexagon.com/products/product-groups/embedded-electronics

Figura 2 – Display Ti7 instalado na cabine de um trator.

Fonte: Documentação interna da empresa.

equipamento que pode ser instalado em um trator e que possibilita a execução de uma atividade específica em campo, como por exemplo, pulverização, fertilização ou subsolagem.

A Figura 3 apresenta uma ECU da Hexagon para implementos do tipo pulverizador. Já a Figura 4 apresenta um sistema de piloto automático da Hexagon, o *AgrOn Track Controller*.

Figura 3 – ECU da Hexagon para pulverizadores.

Fonte: (HEXAGON, c2022c)

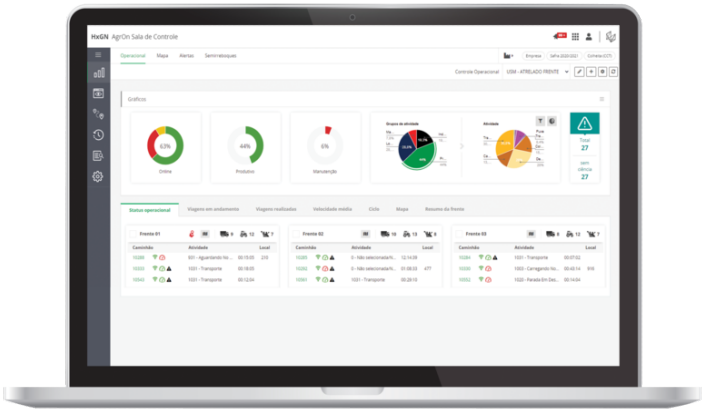
Além dos produtos de eletrônica embarcada para controle da operação de máquinas agrícolas, a Hexagon também comercializa softwares de planejamento e otimização de plantio, e oferece uma plataforma para o monitoramento de máquinas que possuem os displays instalados, chamada Sala de Controle ou *Control Room*. A Figura 6 apresenta uma visão da Sala de Controle.

Quanto à divisão de atribuições entre escritórios: a produção, assim como os times de desenvolvimento de software embarcado, hardware e parte da equipe de desenvolvimento web, se concentram na sede de Florianópolis, sendo o escritório de

Figura 4 – Sistema da Hexagon para piloto automático de tratores.

Fonte: (HEXAGON, c2022e)

Figura 5 – Visão da Sala de Controle.



Fonte: Documentação interna da empresa.

Ribeirão Preto responsável por centralizar o suporte a clientes corporativos e pela manutenção e desenvolvimento de produtos como a Sala de Controle ou os softwares de planejamento e otimização. O escritório de Madrid é um ponto de suporte e assistência técnica para os clientes da Europa, Oriente Médio e África.

Os principais clientes da empresa podem ser divididos pelos tipos de mercado em que atuam:

- Agrícola: Grupo São Martinho e Raízen, que trabalham com o cultivo de cana-de-açúcar e produzem açúcar, etanol e energia elétrica a partir do bagaço residual.
- Florestal (silvicultura): Suzano e Cenibra, que atuam na produção de papel e celulose a partir de madeira de eucalipto.
- *Aftermarket*: Frendt e Razera, revendedores de implementos e peças de reposição para máquinas agrícolas.

- OEM: Jan, Jacto e PVT, fabricantes de equipamentos agrícolas que comercializam os produtos Hexagon para operação integrada com seus próprios produtos.

2.1.3 Modelo de negócio

O modelo de negócio da Hexagon consiste na entrega de um pacote de soluções customizável, do qual o display é o componente central. De acordo com a necessidade de cada cliente, funcionalidades são liberadas por meio de "ativações", em um sistema similar ao de licenças de software. Por exemplo, um cliente que deseja fazer uso do piloto automático da Hexagon precisa comprar, além do display e do sistema *Track Controller*, a ativação para essa funcionalidade. Devido à natureza sazonal das operações no campo, com disponibilidade de tempo limitada para a execução das atividades, além da forte dependência de condições climáticas favoráveis e do alto custo de manutenção das máquinas agrícolas, a disponibilidade e eficiência do suporte técnico se tornam atributos estratégicos para os negócios da Hexagon.

2.1.4 Atribuições da equipe de suporte

O suporte da Hexagon é dividido em vários níveis: primeiramente, é oferecido suporte a nível de campo, sendo prestado pelos técnicos que acompanham as instalações dos clientes. As solicitações de suporte que não podem ser atendidas completamente pelos técnicos podem ser encaminhadas à assistência técnica, caso sejam relacionadas a trocas e defeitos de hardware, ou direcionadas à equipe de serviços de Ribeirão Preto, onde são filtradas por produto. Solicitações relacionadas à Sala de Controle e outros produtos de responsabilidade de Ribeirão Preto são tratadas pelo time de desenvolvimento da unidade, enquanto as demais são encaminhadas para Florianópolis, e recebidas pela equipe de suporte.

Nesse contexto, a equipe de suporte de Florianópolis tem como principal atribuição organizar as solicitações recebidas, realizar uma análise preliminar e encaminhar os resultados às respectivas equipes de interesse dentro da unidade, alinhando prioridades com os líderes de cada equipe e mantendo contato com o time de serviços de Ribeirão Preto ou, caso necessário, com o próprio cliente, para compreender melhor as circunstâncias de cada solicitação, e eventualmente requisitar informações ou testes adicionais.

Outras responsabilidades da equipe de suporte são:

- Apoio às equipes de desenvolvimento com a realização de investigações relacionadas aos pedidos de suporte;
- Alterações e/ou correções nos arquivos de configuração dos computadores de bordo de cada cliente;

- Automatização de tarefas internas repetitivas por meio de *scripts* que são posteriormente disponibilizados para membros de outras equipes;
- Acompanhamento do processo de validação em campo de novas soluções, realizando as configurações e testes necessários.

A equipe de suporte deve manter-se atualizada em relação a novas funcionalidades, lançamento de versões de software, *bugs* conhecidos e demandas de cada cliente. É necessário também ter familiaridade com cada solução da empresa com médio nível de aprofundamento, bem como conhecer as etapas de configuração envolvidas, fazendo uso da documentação interna elaborada pelos desenvolvedores. Frequentemente, a equipe de suporte também colabora com o processo de documentação, já que depende dela para realizar suas tarefas.

2.1.4.1 Organização das tarefas

A estrutura adotada para organizar o trabalho da equipe é uma adaptação da metodologia *Scrum*, popular em empresas de tecnologia e também adotada pelas equipes de desenvolvimento da Hexagon em todas as unidades. Nesse método, as tarefas são organizadas em blocos de duas semanas, denominados *sprints*. Existem várias reuniões periódicas, chamadas de cerimônias do *Scrum* (DRUMOND, c2022), listadas abaixo:

- No começo de cada *sprint*, é realizada uma reunião entre os membros da equipe chamada *sprint planning*, onde são decididas as atividades referentes ao *sprint* e é feita uma estimativa do tempo e esforço necessários para a conclusão de cada tarefa.
- Diariamente, uma reunião rápida é realizada para que cada membro da equipe reporte que tarefas está executando, qual o andamento delas, e se há impedimentos ou travamentos na execução de alguma tarefa. Essa reunião é chamada de *stand-up*.
- Ao final de cada *sprint*, é realizada uma reunião com todas as equipes da unidade de Florianópolis, chamada *sprint review*, para apresentar as tarefas realizadas durante a *sprint* e seus resultados.
- Com menor frequência (a cada três ou seis meses), é realizada uma reunião onde o desempenho da equipe é avaliado pelos próprios membros, em termos do que está funcionando e o que poderia ser melhorado, chamada de *retrospective*.

Para o gerenciamento de tarefas e *sprints*, é adotada a ferramenta *Jira*[2], da empresa *Atlassian*. Em geral, o *Jira* é voltado para a gestão de projetos, mas a equipe de suporte é interpretada e gerenciada como um projeto dentro dessa ferramenta.

A equipe de suporte não trabalha diretamente com desenvolvimento de software, mas desenvolve alguns *scripts* para uso interno, que são armazenados em um repositório *online* com controle de versão, no serviço *Bitbucket*, também da *Atlassian*.

2.2 Sobre a linha de displays Ti5, Ti7 e Ti10

2.2.1 Hardware

Em termos de especificações gerais de hardware, todos os displays possuem conexão com a *internet* via Wi-Fi ou redes móveis, módulo GNSS para obtenção de posicionamento, tela sensível a toque e entradas USB, CAN (*Controller Area Network*) e serial. Algumas especificações, como tipo de processador, armazenamento, memória RAM e tamanho da tela, conforme comentado anteriormente, dependem do modelo. A Tabela 1 apresenta um resumo das principais diferenças entre os modelos de display.

Tabela 1 – Especificações principais dos displays.

	Ti5	**Ti7**	**Ti10**
Tamanho da tela	5 polegadas	7 polegadas	10,1 polegadas
Processador	ARM Cortex-A9, único núcleo, 800 MHz	ARM Cortex-A9, único núcleo, 800 MHz	ARM Cortex-A35, quatro núcleos, 1.2 GHz
Memória e armazenamento	RAM: 1 GB DDR3 FLASH: 4 GB eMMC	RAM: 1 GB DDR3 FLASH: 4 GB eMMC	RAM: 2 GB DDR3 FLASH: 32 GB eMMC
GNSS	Sim	Sim	Sim
Resolução	800×480	800×480	1280x800 (HD)
Wi-Fi	Não	Opcional	Sim
Rádio	Não	Opcional	Opcional
Redes móveis	2G/3G/4G	2G/3G/4G	2G/3G/4G/5G
Bluetooth	Não	Opcional	Bluetooth 5.0
Interfaces	USB (x1), CAN (x1) e RS-232 (x1)	USB (x1), CAN (x2) e RS-232 (x2)	USB (x2), CAN (x3) e RS-232 (x2)

Fonte: HEXAGON (c2022d)

Os displays Ti5 e Ti7 possuem especificações muito similares, diferindo apenas em relação ao tamanho da tela e alguns componentes opcionais. Já o Ti10 tem um processador com arquitetura diferente, o dobro de memória, e o quádruplo de armazenamento.

Alguns dos módulos de hardware dos displays são fabricados por terceiros. Por exemplo, o módulo GNSS é fabricado pela empresa *Novatel*, e existem diferentes tipos de modem para redes móveis, alguns produzidos pela empresa *Sierra Wireless*, outros pela empresa *Telit*. A existência de hardware de terceiros adiciona uma etapa adicional no processo de suporte aos displays, na qual é necessário estabelecer comunicação com o fabricante quando existem solicitações de suporte envolvendo esses módulos. Frequentemente, os fabricantes solicitam a coleta de *logs* de depuração, e outras informações sobre o *status* de operação dos módulos.

[2] https://www.atlassian.com/br/software/jira

Existem pelo menos sete modelos diferentes de modem que podem integrar os displays, e um número similar de módulos GNSS, além de recursos opcionais para aumentar a acurácia do posicionamento, que podem ser contratados pelos clientes. Essa modularidade, em adição às arquiteturas diferentes dos processadores, representa um desafio considerável para os times de desenvolvimento de software embarcado, que devem construir aplicações que sejam compatíveis com todas as variantes possíveis.

2.2.2 Sistema operacional

O sistema operacional dos displays é uma distribuição Linux customizada, baseada no *Yocto Project*[3], um projeto da Linux Foundation[4] voltado para sistemas embarcados e IoT (*Internet of Things*, Internet das coisas (RED HAT, 2019)). O *Yocto Project* foi lançado em 2011 com a colaboração de 22 empresas e organizações do ramo de software de código aberto (THE LINUX FOUNDATION, c2011). Neste documento, o sistema operacional do display será chamado simplesmente de *Yocto*[5].

O processo de criação de uma distribuição customizada do *Yocto* inicia a partir de um repositório padrão, chamado *poky*, que contém as dependências básicas para o sistema operacional. Novos pacotes de software podem ser acrescentados ao *Yocto* a partir de *layers* (camadas). Uma *layer* contém um conjunto de instruções, chamadas de *recipes* (receitas), que determinam quais tarefas serão executadas durante o processo de compilação. A ferramenta de compilação do *Yocto* se chama *bitbake*.

Um exemplo de uso de *layers*: para que o sistema operacional tenha uma versão específica de Python, como a versão 3.8, é necessário adicionar a *layer* correspondente a essa versão à lista de arquivos utilizada pelo *bitbake*. Portanto, desenvolvedores de software que desejam que suas aplicações façam parte do *Yocto* precisam configurar e disponibilizar as *layers* adequadamente para o projeto.

Ao final do processo, são gerados binários que integram a imagem do sistema operacional, e um conjunto de arquivos utilizado para a compilação cruzada de novos programas, denominado SDK (*Software development kit*, conjunto de desenvolvimento de software).

A Figura 6 ilustra o processo de compilação.

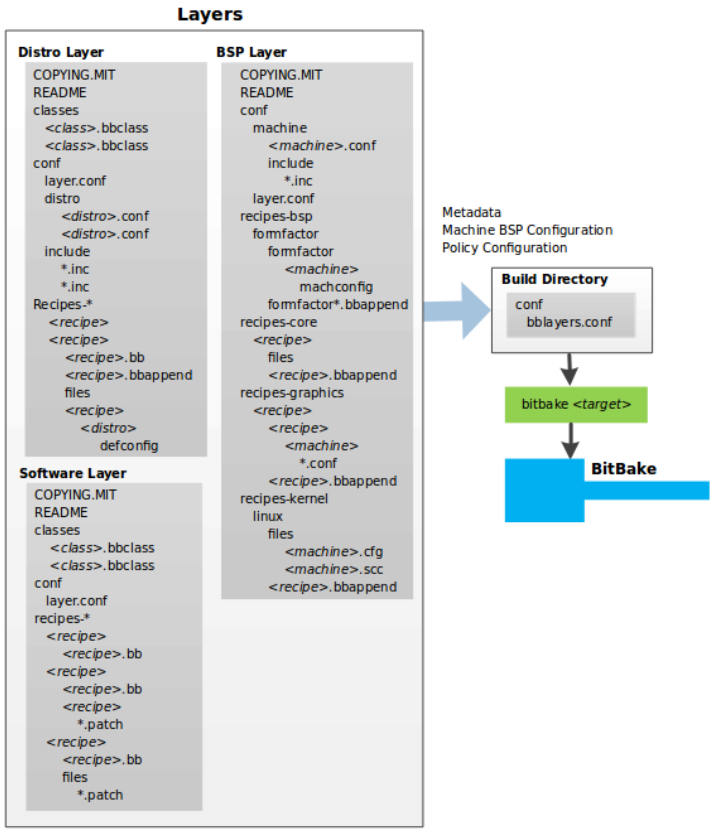
---

[3] https://www.yoctoproject.org/

[4] https://www.linuxfoundation.org/

[5] *Yocto* é o nome do projeto, e cada versão do sistema operacional lançada desde 2011 possui seu próprio nome. Os modelos Ti5 e Ti7 utilizam a versão *rocko*, e o Ti10 utiliza a versão *zeus*. No entanto, o uso de nomes diferentes para cada arquitetura pode tornar o texto confuso, motivo pelo qual essa escolha foi realizada.

Figura 6 – Estrutura utilizada para o desenvolvimento de sistemas baseados no *Yocto Project*.



Fonte: (THE LINUX FOUNDATION, c2018)

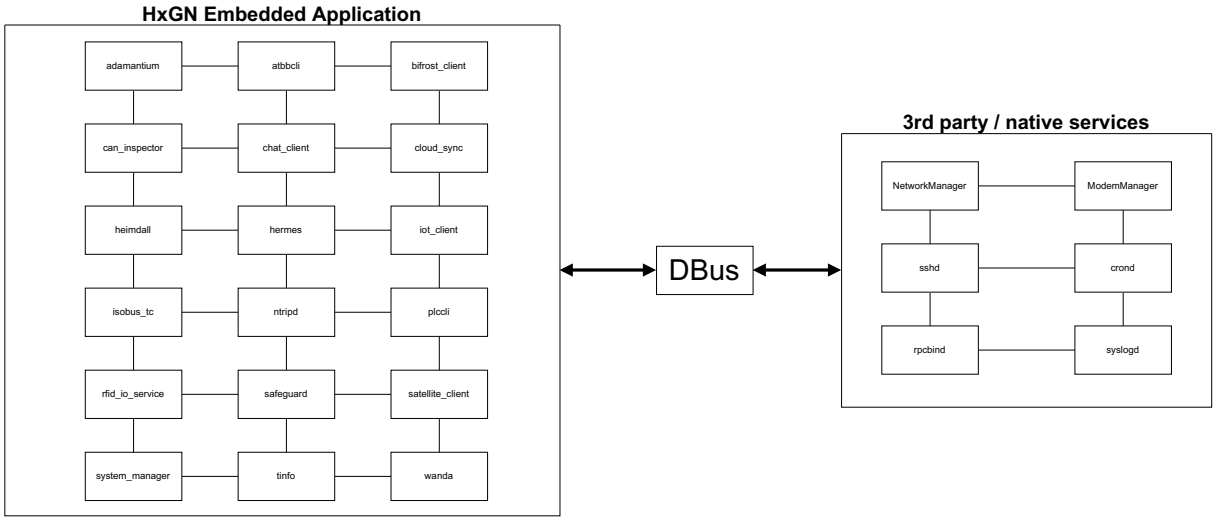
2.2.3 Software embarcado da Hexagon

O processo de desenvolvimento de software da Hexagon utiliza o SDK obtido após a geração da imagem do *Yocto*. As aplicações desenvolvidas para o display são executadas como serviços distintos, e são gerenciadas pelo *systemd*, sistema de inicialização nativo do *Yocto Project* e da maioria das distribuições Linux. A Figura 7 apresenta um diagrama com os serviços que integram a principal aplicação de software embarcado do display.

Atualizações de software são incrementais e disponibilizadas em um arquivo com extensão .uti, contendo apenas os binários das aplicações da Hexagon. A imagem do *Yocto* só pode ser atualizada utilizando um formatador, que consiste em um *flash drive* inicializável, gravado para esse propósito.

A comunicação entre os serviços internos do display é realizada pelo barramento *DBus (Desktop Bus)*. Cada serviço pode ter um conjunto de métodos que são visíveis para os outros processos do sistema, podendo ser invocados para a troca de mensagens entre os processos.

Figura 7 – Principais serviços de software embarcado do display.



Fonte: Documentação interna da empresa.

2.2.4 Arquivos de configuração

A interface do display permite a customização das funcionalidades, o que implica na geração de arquivos de configuração que armazenam as escolhas feitas pelo usuário. Há também parâmetros que não são acessíveis pela interface, mas que se fazem necessários para a operação. Em alguns casos, mesmo quando existe uma interface disponível para alteração dos parâmetros, é interessante que a configuração de alguns serviços seja realizada sem a intervenção do usuário, para evitar erros operacionais. Alguns exemplos de configurações que podem ser realizadas são listados abaixo:

- Periodicidade do envio de arquivos de operação para a nuvem;
- Parâmetros das redes móveis, como APN *(Access Point Name)*, usuário e senha;
- Controle de atualizações de software automáticas.

O gerenciamento dos arquivos de configuração é dividido por cliente, e está associado ao uso da Sala de Controle, que permite ao usuário gerar um pacote com extensão .cti, contendo todos os arquivos de configuração para um cliente específico, que pode ser instalado no display remotamente, ou copiado para um *flash drive* e instalado manualmente pela interface do sistema.

2.2.5 Arquivos de diagnóstico

Quando ocorre alguma falha nos displays em campo, seja relacionada a hardware, software embarcado, ou arquivos de configuração, e uma solicitação de suporte é encaminhada à Hexagon, a primeira evidência que é requisitada para o prosseguimento da investigação é um arquivo de diagnóstico, que consiste em um conjunto de logs e arquivos de configuração que representam o estado atual do display, e podem ser úteis para a resolução do problema. O diagnóstico pode ser solicitado pela interface do display, utilizando um *flash drive* para coleta do arquivo.

Em geral, o diagnóstico é essencial para a investigação de problemas nos displays, especialmente aqueles relacionados a software ou configuração, mas há casos em que seu uso não é suficiente, dependendo da natureza do problema. Problemas relacionados a GNSS e redes móveis são desafiadores para a equipe de suporte, porque os logs disponíveis para esses serviços são limitados, e a reprodução do problema pode estar associada a um conjunto de condições muito específico (operação em uma área remota durante um período de condições climáticas desfavoráveis, o que pode causar alterações no funcionamento do GNSS, por exemplo). Nesses casos, costuma-se solicitar um acesso remoto ao display afetado para realizar testes e aprofundar a investigação.

2.3 Motivação para o acesso remoto

De acordo com informações extraídas da base de dados interna da Hexagon, cerca de 3000 displays estão em operação atualmente em clientes nacionais ou internacionais. O modelo de operação no mercado agrícola e silvícola é sazonal, ou seja, existem períodos específicos para colheita e plantio, que constituem o que é conhecido como safra, e um período intermediário chamado de entressafra, onde são realizadas atividades preparatórias até o reinício do ciclo. O pico das atividades no campo ocorre, naturalmente, durante a safra, onde é desejado que as máquinas agrícolas trabalhem sem interrupções e com a máxima performance possível.

Os períodos de safra e entressafra variam dependendo do tipo de cultura sendo realizada e da região geográfica. No Brasil, a safra de cana de açúcar ocorre entre os meses de abril e novembro, para a Região Centro-Sul, e entre novembro e abril, para a Região Nordeste (TEIXEIRA, c2022).

Do ponto de vista da equipe de suporte, a safra é um período de sobrecarga, e as solicitações de suporte realizadas nesse período são normalmente mais impactantes e mais delicadas do que durante a entressafra, já que cada minuto que uma máquina deixa de operar em campo pode incorrer em prejuízo considerável ao cliente.

Se o problema encontrado for relacionado ao hardware do display, os clientes costumam optar pela troca imediata, pois existem unidades de reserva na maioria dos

casos. Entretanto, problemas que são classificados como *bugs* podem acometer uma frota inteira e inviabilizar a troca de todos os displays envolvidos, e o envio dos produtos afetados para a assistência técnica, que está situada em Florianópolis, oferece um desafio de logística, o que pode atrasar o processo de suporte.

Além disso, mesmo que o time de QA (*quality assurance*) da Hexagon tenha a sua disposição diversos displays para testes e reprodução de *bugs* encontrados em campo, alguns casos são verificados apenas em condições muito específicas de uso, que só podem ser reproduzidas no mesmo ambiente de operação.

Essa é uma descrição não-exaustiva dos fatores que levaram à implantação de um sistema de acesso remoto aos displays da Hexagon. Dois exemplos de situações encontradas pela equipe de suporte que foram solucionadas após sessões de acesso remoto são descritos abaixo.

2.3.1 Exemplo 1: Falha de comunicação com o módulo GNSS

Em março de 2022, clientes do *aftermarket* na Europa reportaram casos em que um alarme de falha de comunicação com o módulo GNSS aparecia na tela dos displays. Essa é uma situação crítica para a operação, já que todos os serviços da Hexagon dependem da informação de posicionamento do display. Esse problema estava relacionado a uma funcionalidade específica de GNSS, da qual o cliente era assinante. Quanto o cliente tentava utilizar essa funcionalidade, o alarme era exibido, e o GNSS só voltava a operar corretamente após desativá-la.

Ao receber a solicitação, a equipe de suporte entrou em contato com o fabricante dos módulos GNSS do display, a empresa *Novatel*, que faz parte da divisão de autonomia e posicionamento da Hexagon. O suporte da *Novatel* emitiu um comunicado informando que módulos GNSS instalados em equipamentos que estavam em operação de 21 de janeiro a 4 de março de 2022 receberam uma mensagem inválida que desconfigurou os módulos e causou o comportamento observado no cliente. O comunicado também informou que os módulos afetados precisariam ser reconfigurados utilizando comandos de terminal para que o problema fosse resolvido. Após receber as informações da *Novatel*, a equipe de suporte da Hexagon agendou uma sessão de acesso remoto com os clientes afetados na Europa para validar que a reconfiguração dos módulos, conforme as instruções do comunicado, resolveria o problema. Com a realização dos testes, o problema foi resolvido e um *script* foi elaborado para automatizar as etapas de reconfiguração dos módulos, que foi disponibilizado aos técnicos responsáveis pelo suporte a nível de campo para os clientes afetados.

2.3.2 Exemplo 2: Problemas de conexão com modem 4G

Um cliente corporativo do ramo da cana-de-açúcar no Brasil reportou que alguns dos displays em campo estavam perdendo a capacidade de reconexão à rede 4G após operar em uma área sem sinal e retornar à area de cobertura, e que apenas após reiniciar os displays o problema era solucionado.

A solicitação de suporte chegou a partir da equipe de serviços de Ribeirão Preto. A primeira tentativa de resolução da equipe foi solicitar que o cliente atualizasse sua versão de software, porém isso não foi suficiente para solucionar o problema. Posteriormente, a equipe de suporte solicitou aos membros da equipe de QA uma tentativa de reprodução do problema, simulando a área de operação sem sinal com a retirada da antena do display. Várias tentativas de reprodução do bug foram realizadas, mas sem sucesso.

Finalmente, a equipe de suporte solicitou uma sessão de acesso remoto com um dos displays afetados. Um membro da equipe de serviços de Ribeirão Preto se deslocou até a localidade do cliente para auxiliar na tratativa. Durante o acesso remoto, vários comandos foram enviados ao modem 4G do display para avaliar o status de operação do mesmo e coletar *logs* de depuração. Verificou-se que o firmware do modem entrava em estado de *deadlock* após a perda do sinal 4G, mas o motivo não pôde ser determinado, e que reiniciar apenas o modem, não todo o display, resolveria o problema temporariamente, até que os logs pudessem ser analisados pelo time de desenvolvimento. Com base nessas informações, a equipe de suporte elaborou um script que verifica periodicamente o status do modem e, em caso de perda do sinal 4G, envia um sinal de *reset* para o modem, o que foi considerado menos disruptivo para a operação do cliente do que reiniciar o sistema completo.

O próximo capítulo descreve as tecnologias e sistemas que possibilitaram as sessões de acesso remoto descritas nos exemplos, e que foram cruciais para a resolução dos problemas reportados.

3 Ferramentas de acesso remoto

Este capítulo apresenta as ferramentas, plataformas e tecnologias utilizadas para realizar o acesso remoto a displays da Hexagon, e uma pesquisa de mercado para investigar quais são as soluções disponibilizadas por competidores nesse contexto.

A seção 3.1 apresenta o AgrOn Acesso Remoto, com informações sobre sua concepção, arquitetura, casos de uso e estado atual da implementação. A seção 3.2 apresenta o processo de acesso remoto aos displays via SSH utilizado pela equipe de suporte, que foi automatizado via script, e alguns exemplos de casos de uso desse script também são expostos, bem como as dificuldades encontradas durante o seu uso e as conclusões que levaram à concepção desse projeto.

Ao final do capítulo, na seção 3.3, encontram-se os resultados de uma pesquisa de mercado realizada para entendimento das soluções de acesso remoto disponíveis atualmente, incluindo testes com a ferramenta *Shellhub*, da *O.S. Systems*[1].

3.1 Sobre o AgrOn Acesso Remoto

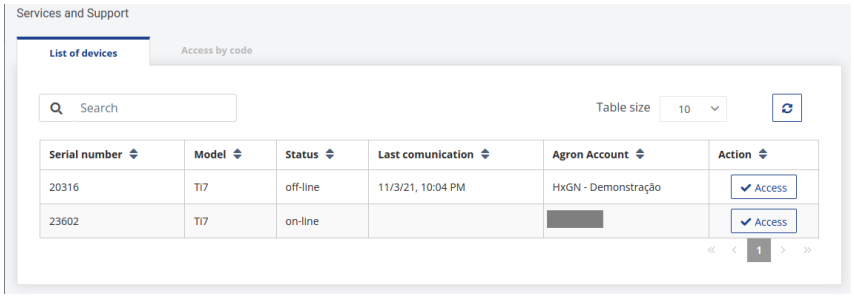
Até janeiro de 2022, o sistema principal de acesso remoto para displays da Hexagon era o *TixViewer*, desenvolvido em Java, cujos desenvolvedores já não fazem parte do quadro de colaboradores da empresa, e que apresentava vários bugs sem resolução há alguns anos. Durante o planejamento estratégico das equipes de desenvolvimento da Hexagon, concluiu-se que tentar realizar a manutenção desse sistema seria mais complexo e custoso do que fazer uma solução completamente nova.

O projeto do *TAgrOn Acesso Remoto* foi conceitualizado em meados de 2021, e o pré-lançamento (interno) ocorreu em janeiro de 2022. Por decisão comercial, foi definido que a licença do AgrOn Acesso Remoto seria disponibilizada também como um produto disponível para os clientes, por representar um diferencial no suporte aos displays. O cliente que deseja adquirir acesso ao sistema deve possuir uma conta no sistema central de acesso aos produtos Hexagon, também chamada de conta AgrOn. A existência de uma conta AgrOn significa que é possível controlar o acesso aos displays realizando associações, porém também existe a opção de acesso por código, onde o usuário digita um código gerado na interface do display e ganha acesso ao mesmo, sem necessidade de associação prévia, o que representa o caso de uso principal, já que durante a concepção do sistema, foi avaliado que o método de código de acesso traria vantagens por não ser necessário realizar alterações no cadastro toda vez que um usuário desejasse acessar um novo display.

A Figura 8 apresenta a interface de seleção de displays do sistema. Já a figura 9 apresenta a visão principal durante a conexão a um display.

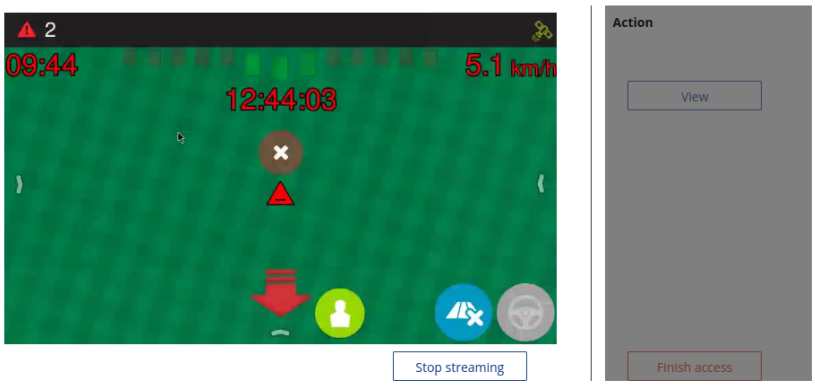
[1] https://www.ossystems.com.br/

Figura 8 – Tela de seleção de displays do AgrOn Acesso Remoto.



Fonte: Documentação interna da empresa.

Figura 9 – Tela de sessão do AgrOn Acesso Remoto.



Fonte: Documentação interna da empresa.

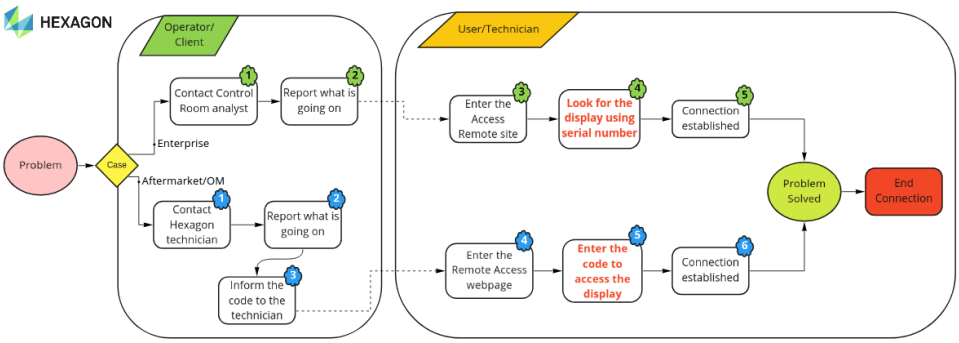
Em termos de arquitetura, o sistema consiste em uma aplicação web, cujo codinome interno é *Vision*, e uma aplicação embarcada responsável pelo controle de conexão e transmissão de dados do lado do display, cujo codinome interno é *Wanda*. Os codinomes serão utilizados para diferenciar as duas partes do projeto ao longo do documento.

As informações fornecidas a respeito do AgrOn Acesso Remoto neste documento foram adaptadas com base na documentação do projeto, disponível na base de conhecimento interna da Hexagon.

3.1.1 Fluxograma do AgrOn Acesso Remoto

A Figura 10 apresenta um fluxograma que contempla os dois casos de uso da ferramenta: acesso por associação e acesso por código.

Figura 10 – Fluxograma de operação do AgrOn Acesso Remoto.

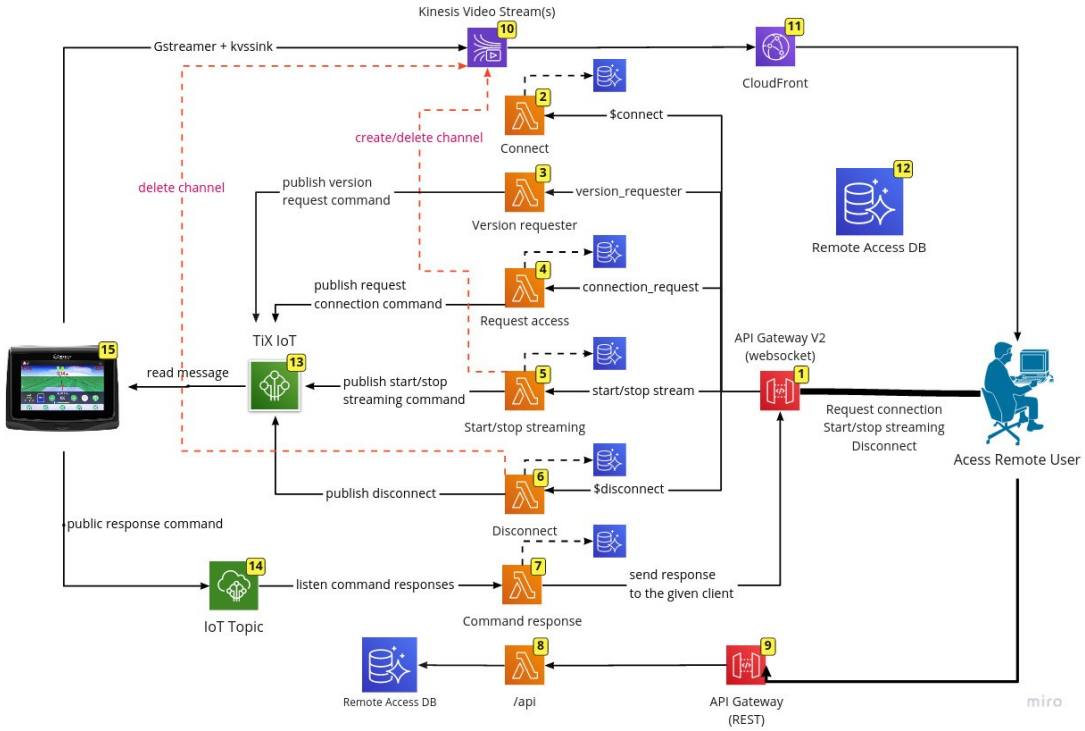


Fonte: Documentação interna da empresa.

3.1.2 Arquitetura da solução

A Figura 11 apresenta um diagrama de arquitetura para o AgrOn Acesso Remoto.

Figura 11 – Diagrama de arquitetura do *Vision*.



Fonte: Documentação interna da empresa.

Os serviços enumerados são detalhados a seguir.

1. API *WebSocket*: conjunto de rotas responsáveis pelo controle de sessão. Cada rota da API possui uma *Lambda* associada à mesma. A escolha de uma API *WebSocket* foi realizada devido à natureza assíncrona da troca de mensagens entre a aplicação web e o display, e também pela possibilidade de comunicação *full-duplex*.

2. Rota *$connect*: rota padrão de APIs *WebSocket*. Quando essa rota é invocada, um novo registro é realizado no banco de dados da aplicação (item 12), em uma tabela no *DynamoDB*.

3. Rota *version_requester*: como o AgrOn Acesso remoto só está disponível a partir de uma certa versão de software, antes de iniciar uma sessão, um comando é enviado via mensagem MQTT para verificar o se o display está online ou offline, e qual a versão de software instalada. A informação contida na resposta do display é então utilizada para proceder com a sessão, ou fornecer um *feedback* ao usuário em caso de falha ou versão incompatível.

4. Rota *connection_request*: responsável por encaminhar, também via mensagem MQTT, um comando de conexão ao display. Também é responsável por realizar a validação da autorização do usuário para acessar o display solicitado, no caso de conexão via associação, ou pela validação do código de acesso.

5. Rotas *start_stream* e *stop_stream*: responsáveis pelo controle do *streaming* de vídeo durante a sessão. Vários usuários podem se conectar ao mesmo display simultaneamente, mas no caso da rota *stop_stream*, o *stream* não é fechado enquanto houver mais de um usuário assistindo à conexão.

6. Rota *$disconnect*: pode ser invocada pelo servidor (display) ou o cliente (interface web). De forma complementar à rota *$connect*, os registros de conexão criados no *DynamoDB* são deletados por essa rota, ao final da sessão.

7. *Lambda command_response*: responsável por escutar o tópico MQTT para o qual os displays enviam as respostas aos comandos (item 14) e redirecionar cada resposta ao *websocket* correto. Nesse contexto, os comandos enviados são relacionados ao fluxo da aplicação, como: conectar, desconectar, iniciar *stream*, informar versão de software, entre outros. Todas as respostas recebidas são armazenadas no banco de dados do AgrOn Acesso Remoto, em uma tabela no RDS.

8–9. API REST: conjunto de *endpoints* utilizados no *front-end* para obter e disponibilizar informações como: lista de displays acessíveis, URL do *stream* da sessão e nível de permissão do usuário.

10. *Kinesis Video Stream*: solução da AWS para gerenciamento de *streaming* de vídeo.

11. *CloudFront*: responsável por redirecionar os dados do *stream* da sessão à página web.

12. Bancos de dados: Um banco de dados relacional, hospedado no RDS, e que contém os dados das associações display/usuário e os comandos enviados ao display e suas respostas; e um não-relacional no DynamoDB, que contém o registro das conexões ativas.

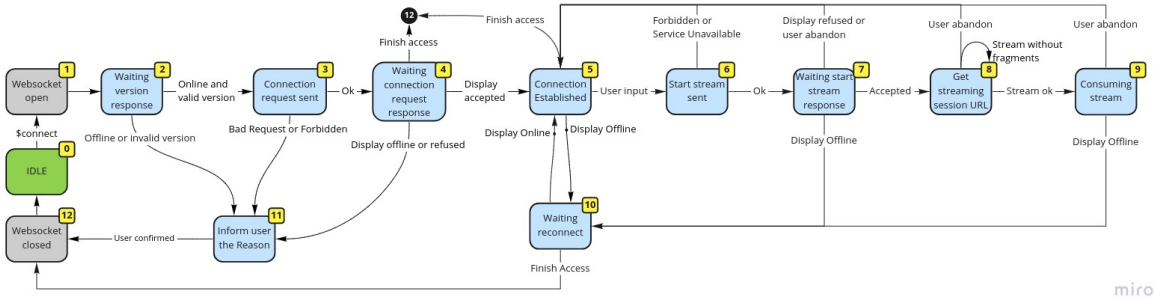
13–14. Tópicos MQTT: para o envio de mensagens ao display e recebimento das respectivas respostas.

15. *Wanda*: serviço embarcado que realiza a interface com o resto do sistema, traduzindo os comandos em ações no display e retornando os resultados.

3.1.3 Diagrama de máquina de estados

A Figura 12 apresenta o diagrama de máquina de estados do AgrOn Acesso Remoto.

Figura 12 – Diagrama de máquina de estados do AgrOn Acesso Remoto.



Fonte: Documentação interna da empresa.

Os estados do sistema são descritos a seguir.

- A operação se inicia quando o usuário selecona um display na lista de displays permitidos (**estado 0**).
- A rota *$connect* da API *WebSocket* é invocada e, quando a conexão é estabelecida (**estado 1**), uma mensagem solicitando a versão de software do display é enviada (**estado 2**).

- Se o display possui uma versão de software compatível, uma mensagem é enviada ao servidor para solicitar o início de uma sessão (**estado 3**). Caso contrário, uma mensagem de *feedback* é exibida ao usuário (**estado 11**) e a conexão é fechada pela rota *$disconnect* (**estado 12**).
- No estado 3, se a conexão é válida (por meio da verificação de associação entre display e usuário), uma mensagem é enviada para o display solicitando a conexão, e o sistema espera a resposta (**estado 4**).
- Se a solicitação de conexão é aceita pelo display, o acesso remoto é estabelecido (**estado 5**). Em caso de falha na conexão ou na validação do código de acesso, o sistema finaliza a sessão de forma análoga ao item anterior (estados 11 e 12).
- Nos estados 4 e 5, o usuário pode encerrar a sessão a qualquer momento pelo botão "finalizar sessão"da interface, o que invoca a rota *$disconnect* (estado 12).
- Com a sessão estabelecida, o usuário pode escolher visualizar a tela do display, o que significa que uma mensagem para criar um novo *stream* é enviada ao *Kinesis* (**estado 6**).
- O servidor pode retornar erro na criação do *stream*, o que faz o sistema retornar ao estado 5. Se a criação for bem-sucedida, o sistema aguarda a resposta do display (**estado 7**).
- No estado 7, se o display recusar a solicitação de *streaming*, o sistema também retorna ao estado 5. Caso contrário, o sistema tenta se conectar à URL do *stream* criado (**estado 8**) e, em caso de sucesso na conexão, começa a consumir o *stream* (**estado 9**).
- Nos estados 7, 8 e 9, o usuário pode cancelar o *streaming* de vídeo a qualquer momento pelo botão "parar *streaming*", e o sistema retorna ao estado 5.
- Nos estados 5, 7 e 9, se o display perder conexão com a rede, o sistema aguarda pela reconexão (**estado 10**). Quando a conexão é reestabelecida, o sistema retorna ao estado 5. Se a conexão não for reestabelecida, ou o usuário desejar cancelar a sessão, o sistema invoca a rota *$disconnect* (estado 12).

3.1.4 Bancos de dados

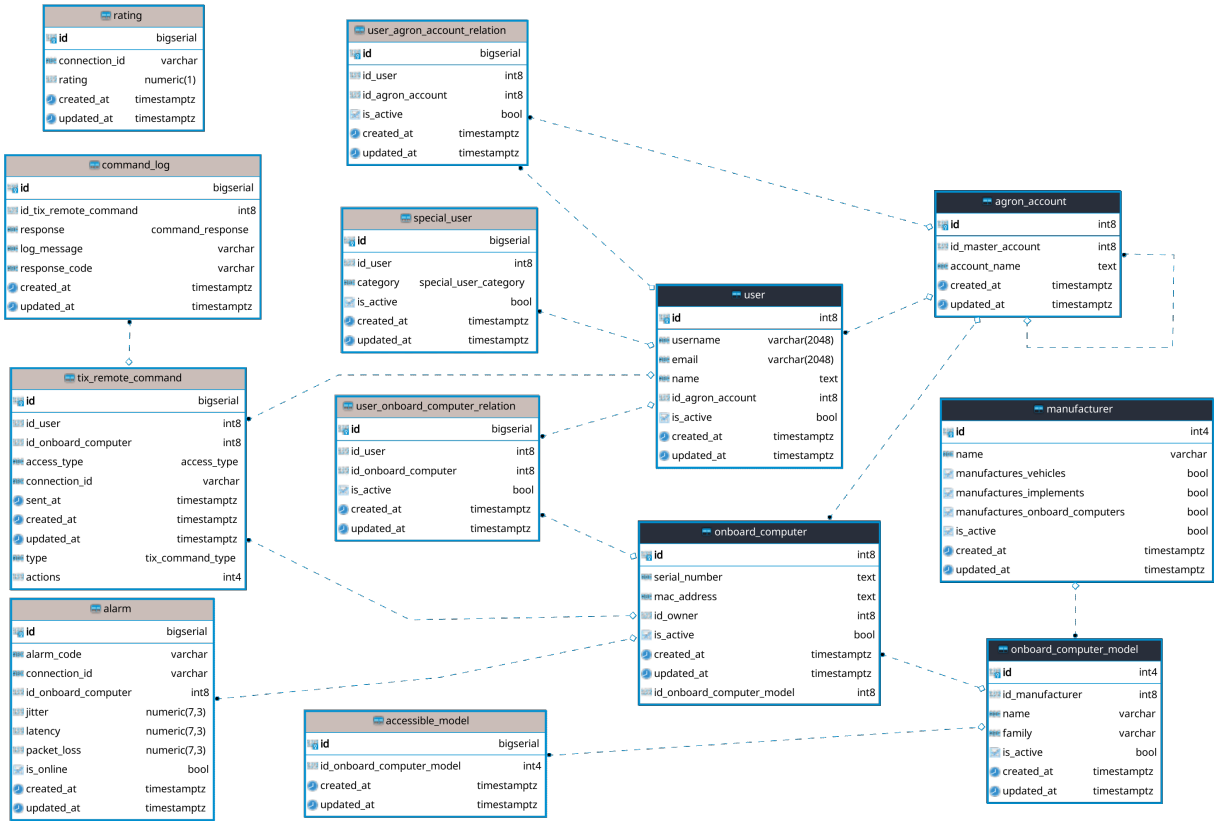
Dois bancos de dados são utilizados no AgrOn Acesso Remoto, sendo um deles relacional, hospedado no RDS, e outro não-relacional, hospedado no DynamoDB. O banco de dados relacional consulta informações de um banco de dados chamado *Atom*, utilizado para o cadastro geral de usuários (contas AgrOn), displays, empresas e outros

recursos utilizados nos serviços da Hexagon, a partir de um serviço de sincronização chamado *Atom Importer*.

A função do banco de dados relacional é armazenar dados relacionados aos comandos enviados para o display e as permissões de cada usuário. As interações entre usuário e sistema são armazenadas para posterior análise de performance da aplicação e auxílio na investigação de problemas.

A Figura 13 apresenta um modelo lógico do banco de dados relacional, separado em dois grupos de tabelas: as tabelas da esquerda são exclusivas ao AgrOn Acesso Remoto, e as tabelas da direita são sincronizadas a partir do *Atom*, conforme citado anteriormente.

Figura 13 – Modelo lógico do banco de dados relacional do AgrOn Acesso Remoto.



Fonte: Adaptado de documentação interna da empresa.

As principais tabelas do banco de dados relacional são descritas abaixo.

- Tabelas *special_user*, *user_agron_account_relation* e *user_onboard_computer_relation*: compõem o esquema de permissões para os usuários do serviço. Os usuários com permissão especial, como administradores, possuem um registro na tabela *special_user*, o que sinaliza que eles podem visualizar todos os displays e alterar a permissão dos demais usuários.

- Tabelas *tix_remote_command* e *command_log*: registram os comandos enviados via API *WebSocket*, e as respostas recebidas.
- Tabela *accessible_model*: lista os modelos de displays disponíveis para acesso.
- Tabela *alarm*: registra o estado da sessão quando um evento inesperado ocorre, como perda de conexão com o display, ou algum erro interno do *back-end*.
- Tabela *rating*: ao final de cada sessão de acesso remoto, o usuário tem a opção de avaliar a qualidade do serviço de uma a cinco estrelas. A avaliação é registrada nessa tabela.

As tabelas do *Atom* estão presentes como uma cópia somente leitura, para manter a consistência dos dados entre os serviços web da Hexagon, sendo elas a fonte de chaves estrangeiras para a maior parte das tabelas do AgrOn Acesso Remoto.

Quanto à tabela criada no *DynamoDB*, sua função é armazenar o status de conexão do display e a lista de usuários acessando cada display (essa informação é obtida pelo número de clientes na API *WebSocket*). Os atributos dessa tabela são descritos abaixo:

- *serial_number*: chave da tabela. É o número de série do display;
- *is_online*: status de conexão do display com os serviços web;
- *disconnected_at*: quando o display perde conexão com o serviço, esse campo é criado com a *timestamp* de desconexão;
- *active_connections*: lista IDs de conexão, que representam os usuários conectados ao display. IDs de conexão são adicionados pela rota *$connect* da API *WebSocket*, e removidos pela rota *$disconnect*.

Todos os componentes da arquitetura web do AgrOn Acesso Remoto dependem da comunicação com o display, que fornece os *frames* de vídeo que são apresentados na tela durante a sessão. A próxima sub-seção apresenta algumas informações sobre o agente embarcado que é executado no display e completa o fluxo de dados do sistema.

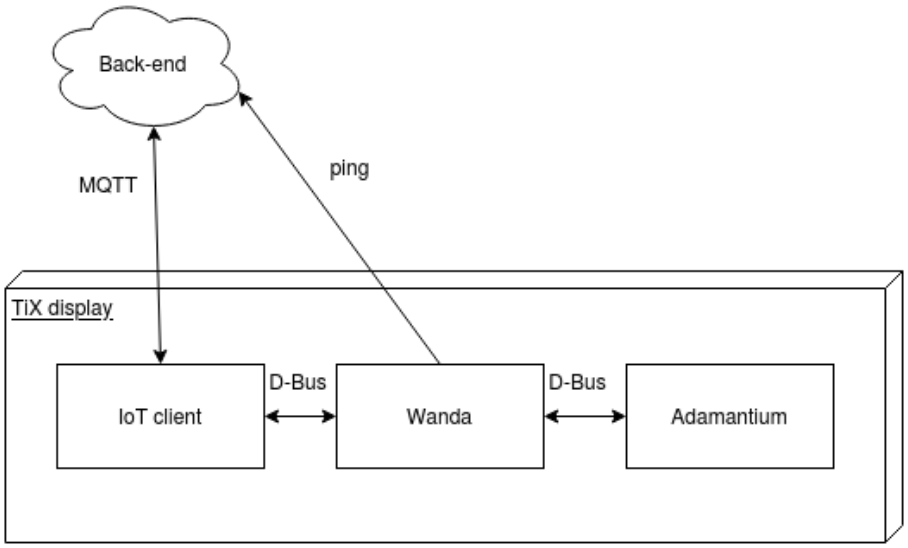
3.1.5 Agente embarcado: *Wanda*

Wanda é um serviço gerenciado pelo *systemd* que é parte integrante da solução de software embarcado da Hexagon para os displays. Esse serviço provê uma interface *DBus* para troca de mensagens com outros serviços embarcados, em específico o *adamantium* e o *iot_client*, apresentados no capítulo 2.2, além de possuir um

mecanismo de *keep-alive* para a conexão de acesso remoto, e se comunicar com o *back-end* via mensagens JSON.

A Figura 14 apresenta o diagrama de *deployment* do *Wanda*.

Figura 14 – Diagrama de *deployment* do *Wanda*.



Fonte: Documentação interna da empresa.

As principais interações do *Wanda* com os demais serviços, incluindo o *back-end*, são descritas abaixo.

- *Adamantium*: A comunicação do *Adamantium* com o *Wanda* ocorre via *DBus*. Resumidamente, a interface disponibiliza um método para a obtenção do código de acesso do display (que é exibido pelo *Adamantium*) e um sinal que indica se há uma sessão de acesso remoto ativa no momento, que é monitorado pelo *Adamantium* para controlar a exibição de um ícone na tela de operação, indicando a existência desse acesso.

- *IoT Client*: ao iniciar o sistema, o *Wanda* se comunica via *DBus* com o *iot_client* para registrar os tópicos MQTT utilizados pelo *Vision* para a comunicação entre o *back-end* e o display. Cada tópico possui um método *handler* implementado na *DBus*; desta forma, sempre que uma nova mensagem chegar em um tópico, o *iot_client* sinaliza o recebimento para o *Wanda* e invoca o respectivo método para o tratamento dessa mensagem.

- Mecanismo de *keep-alive*: consiste em um sistema de mensagens*ping/pong* entre o *display* e o *back-end*. As mensagens de *ping* contém estatísticas de

rede e são enviadas periodicamente, em um intervalo configuável, cujo padrão é 60 segundos. Quando uma mensagem de *pong* não é recebida antes do próximo *ping*, o sistema toma ações para garantir o reenvio das mensagens ao display.

3.1.6 Estado atual da implementação

Atualmente, o AgrOn Acesso Remoto conta apenas com a funcionalidade de *streaming* da tela do display disponível em ambiente de produção. No entanto, uma funcionalidade complementar, que permite ao usuário controlar a tela do display remotamente, está sob testes e disponível no ambiente de desenvolvimento.

Tanto o AgrOn Acesso Remoto como o *TixViewer*, sistema legado da Hexagon para o mesmo propósito, nunca implementaram o acesso ao terminal, sendo necessário o uso de outras ferramentas para esse propósito. O acesso ao terminal é necessário porque nem todos os parâmetros e opções de configuração estão disponíveis na interface do display. Os exemplos descritos nas seções 2.3.1 e 2.3.2 são casos em que apenas o acesso à tela, mesmo com a possibilidade de interação, não seria suficiente. A próxima seção descreve as ferramentas utilizadas pela equipe de suporte da Hexagon para realizar o acesso remoto ao terminal dos displays.

3.2 Acesso remoto via SSH

SSH é um protocolo de rede do tipo cliente/servidor, baseado em TCP/IP, e popularmente utilizado para o acesso remoto de usuários a computadores. A característica principal do SSH é a segurança e confiabilidade, com camadas de autenticação, criptografia e verificação de integridade dos dados (BARRETT; SILVERMAN, 2001).

Um servidor SSH pode ser um serviço, como o *sshd* do Linux, que implementa o protocolo e é responsável por aceitar ou rejeitar conexões de entrada. Esse serviço normalmente é executado no computador que deseja-se acessar remotamente. Já um cliente SSH é um programa executado externamente, seja no computador do usuário ou em um ponto de infraestrutura de rede, que envia requisições ao servidor. Essas requisições podem ser comandos de terminal, ou envio/recebimento de arquivos.

A autenticação via SSH ocorre pela verificação de senha. O usuário (cliente) que deseja se conectar ao computador (servidor) precisa saber a senha gerada pelo serviço de SSH do mesmo para efetuar o acesso remoto. Existe também a possibilidade da geração manual de um par de chaves pública/privada, onde a chave pública é instalada no servidor, e a privada permanece no cliente, o que elimina a necessidade da senha, pois no momento da conexão, o serviço de SSH consegue verificar a identidade do cliente porque as mensagens enviadas pelo protocolo são criptografadas com a chave pública, e descriptografadas com a chave privada.

Para realizar o acesso, o usuário precisa saber o IP da máquina que deseja

acessar, o usuário e a senha de SSH. No contexto de uma rede local, saber o IP da máquina de destino é relativamente simples. Já no caso de máquinas em redes diferentes, surgem complicações, sendo as mais comuns listadas abaixo:

- *Firewall*: normalmente, conexões de entrada à porta utilizada pelo serviço de SSH, que por padrão é a porta 22, são bloqueadas. A maioria das infraestruturas de rede, tanto corporativas quanto domésticas, utiliza um *firewall* para controlar o acesso de agentes externos à rede local.
- *NAT* (*Network Address Translation*): um recurso utilizado em roteadores para mapear IPs de rede local a um IP público, de forma que não sejam alocados múltiplos IPs públicos para cada máquina, conservando o espaço de endereços da Internet (WING, 2010). Isso significa que muitas vezes é impossível determinar o IP público de uma máquina específica em uma rede externa, mesmo que ela esteja conectada à Internet, porque a presença do NAT faz com que conexões externas sejam sempre endereçadas ao IP do roteador, que se responsabiliza por mapear as requisições para os devidos endereços de IP locais, mapeamento este que não é transparente.
- *Proxy*: especialmente comum em redes corporativas, o *proxy* é um filtro utilizado para restringir os tipos de requisição ou protocolos permitidos em uma rede. O uso de um *proxy* significa que conexões de entrada devem originar apenas de endereços permitidos pelo mesmo, e a situação é análoga para conexões de saída.

A Figura 15 apresenta um diagrama que exemplifica o cenário mais comum que ocorre quando um membro da equipe de suporte tenta acessar remotamente o terminal de um display da Hexagon via SSH.

Algumas estratégias podem ser utilizadas para contornar as dificuldades citadas. Uma delas é o conceito de túnel SSH reverso, que será explorado abaixo.

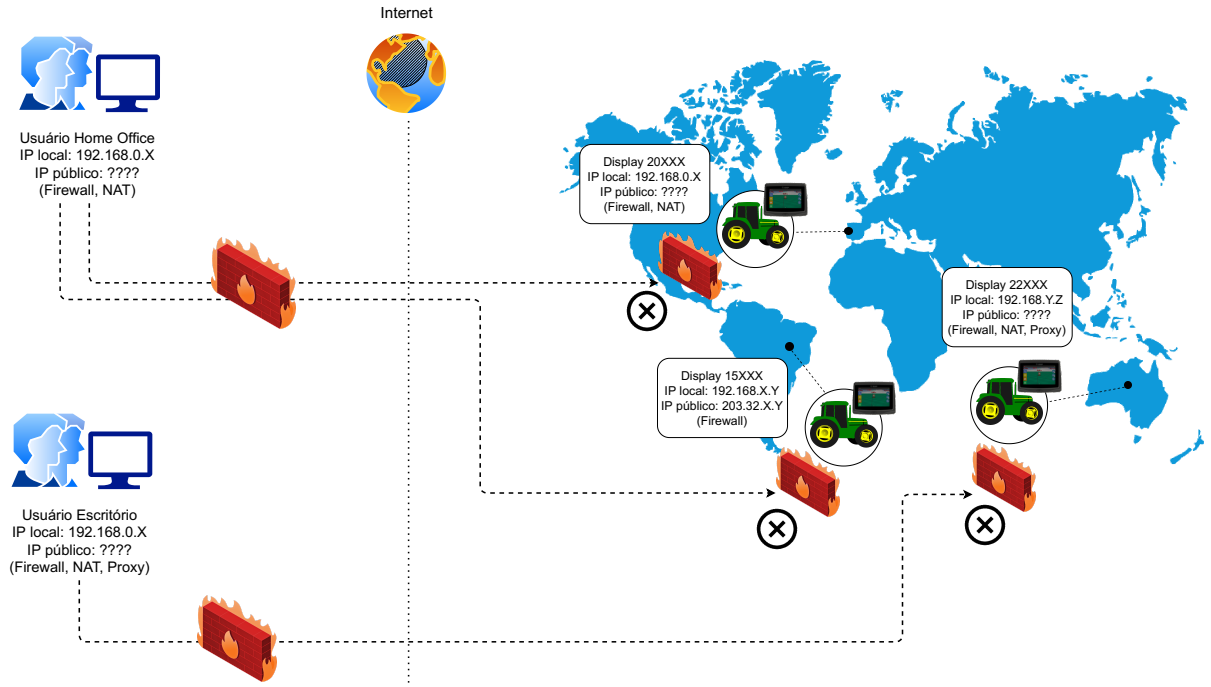
3.2.1 Túnel SSH reverso

Essa estratégia assume a existência de um servidor intermediário, com um IP público único e conhecido (NAT desativado), em uma rede cujo *firewall* foi configurado para permitir conexões SSH em uma porta específica e, consequentemente, as configurações de *proxy* permitem conexões de entrada e saída com endereços de IP externos à rede.

Se um par usuário/computador tem acesso ao servidor intermediário (por meio de seu IP público), o acesso remoto se torna possível com as seguintes etapas:

- O computador se conecta ao servidor intermediário, na porta configurada para aceitar conexões de entrada SSH;

Figura 15 – Cenário de acesso remoto a displays da Hexagon.



Fonte: Arquivo pessoal.

- O computador especifica que todo o tráfego de dados pela porta de SSH será redirecionado para uma porta arbitrária no servidor intermediário;
- O usuário se conecta ao servidor intermediário, utilizando a mesma porta que o computador configurou o redirecionamento do tráfego de dados via SSH.

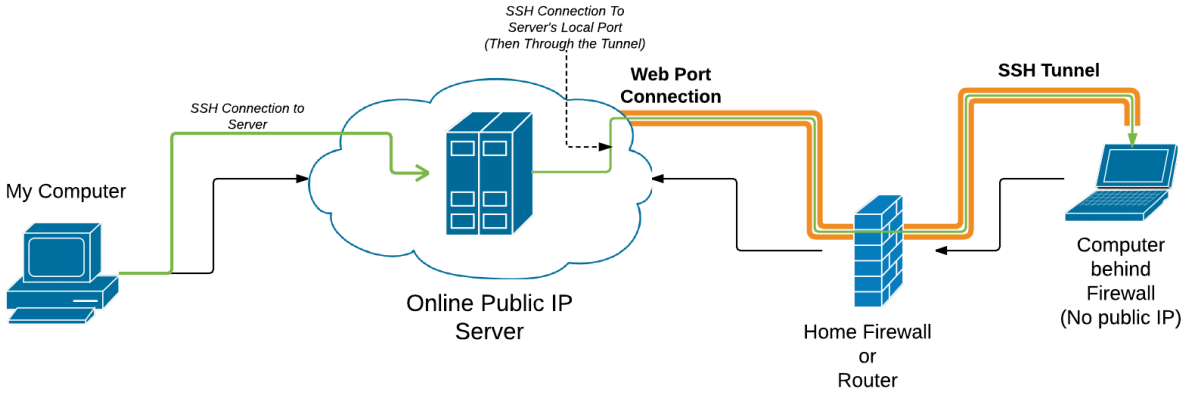
O acesso ainda depende da autenticação bem-sucedida do usuário, que pode ser feita das formas mencionadas anteriormente (senha ou chave pública). A Figura 16 ilustra essa estratégia.

Para automatizar a criação de um túnel SSH reverso e possibilitar o acesso aos displays ds Hexagon, criou-se o script *ReverseSSH*, cujo funcionamento básico é descrito a seguir.

3.2.2 Sobre o script *ReverseSSH*

Originalmente elaborado como um script *shell*, a versão atual do *ReverseSSH* foi escrita em Python por membros da equipe de suporte. Os argumentos de entrada são o número de série do display a ser acessado, a porta a ser utilizada para redirecionamento e algumas *flags* opcionais, como modo de operação em batelada (*batch*) e acesso em *home office* (fora da rede da empresa). O *ReverseSSH* implementa o túnel SSH reverso descrito na seção anterior, utilizando um servidor da Hexagon com

Figura 16 – Exemplo de aplicação de um túnel de SSH reverso.



Fonte: (HIMMELWRIGHT, 2017)

IP público fixo, configurado para este propósito. O script também consulta um dos bancos de dados internos da Hexagon para obter dois parâmetros de hardware dos displays, que são registrados quando eles saem de fábrica, e que servem de entrada para o algoritmo que gera a senha de SSH. Desta forma, possuindo uma cópia do programa que implementa o algoritmo de geração, é possível gerar a senha de SSH para qualquer display, desde que os parâmetros estejam corretos.

A Figura 17 apresenta um trecho da documentação do *ReverseSSH*.

Figura 17 – Trecho da documentação do *ReverseSSH*.

```
## Usage
There are three basic ways to use reverse_ssh: for a single TiX, for a list of them and just to generate/update the `equipments.db`.

Note: the port parameter is mandatory and its value can be any higher than 3000.
#### Single TiX
To run for a single TiX, just run the following command. This will leave you a prompt to run commands and its outputs will be shown at the screen.

`python3 reverse_ssh.py -p 5432 -s 12345`

#### Multiple TiX
To run for multiple displays, you will have to change `serial_list.csv` file (which is located in the same directory as the script) putting the serial numbers
separated by line breaks, keeping the `"serial_number"` header line. Then, just run the following command:

`python3 reverse_ssh.py -p 5432`

Note: there is no problem to have trailing white-spaces in `serial_list.csv` file.
#### Generate equipments.db
To generate/update `equipments.db` simply run this command (port is not necessary as it will not access any TiX):

`python3 reverse_ssh.py --generate`

#### Predefined commands
It is possible to define a list of commands to be run in every TiX when it is accessed (single or multiple). To use this, write the desired commands in
`commands.txt` separated by a line break. An example of commands is shown below.

```
echo 'This is an example of usage.'
echo 'Hello World!'
download /home/titanium/periodic/titanium.log
upload script.sh /home/titanium/sporadic
```
```

Fonte: Arquivo pessoal.

No caso de uso padrão, o script automatiza os seguintes passos:

- Envio de mensagem para um tópico MQTT que o display está escutando por padrão (registrado no serviço *iot_client*);
- Invocação do método *ReverseSSH*, registrado na *DBus* dos displays, com os parâmetros recebidos na mensagem enviada para o tópico MQTT. Esse método é responsável pelo estabelecimento do túnel SSH;
- Geração automática da senha de SSH do display, com os parâmetros obtidos no banco de dados pelo script;
- Criação de conexão SSH entre o computador do usuário e o servidor intermediário, na porta definida como parâmetro.

3.2.3 Dificuldades encontradas no acesso remoto

Os principais pontos de melhoria no gerenciamento do acesso remoto aos displays no modo de terminal são identificados abaixo, e embasam o desenvolvimento deste projeto.

3.2.3.1 Problemas de portabilidade

O script é executado localmente e, para que a execução seja bem-sucedida, as variáveis de ambiente precisam ser configuradas corretamente, como: credenciais de serviços da AWS (para enviar a mensagem ao tópico MQTT), senha de SSH do servidor central e do servidor onde o banco de dados consultado pelo script fica hospedado, e o IP público do servidor central. Além disso, o script depende de várias bibliotecas para Python com versões específicas, e só foi testado e validado em sistemas baseados em Unix.

O interesse em compartilhar a ferramenta com outros times da Hexagon, incluindo a equipe de serviços de Ribeirão Preto, que utiliza primariamente Windows, motiva iniciativas de melhoria na portabilidade do *ReverseSSH*, incluindo a possibilidade de conteinerização, prática comum na distribuição de aplicações multi-arquitetura (MCCARTY, 2020). Outra possibilidade seria transformar o *ReverseSSH* em uma aplicação *serverless*, utilizando a AWS e removendo as restrições de ambiente.

3.2.3.2 Senha de SSH incorreta

Conforme comentado anteriormente, a senha de SSH que é gravada nos displays quando eles saem de fábrica é gerada por um programa auxiliar que usa como parâmetros duas informações associadas ao hardware instalado nos displays: o ID placa base e o MAC address da primeira interface de rede. Essas informações ficam salvas em um banco de dados no servidor central, que é consultado para gerar a senha correta durante a conexão.

Infelizmente, a senha que está gravada no display frequentemente não condiz com a senha gerada durante a tentativa de conexão. Dois dos principais motivos são listados abaixo:

- Quando um dos componentes de hardware do display é trocado pela assistência técnica, parte do processo de troca consiste em validar o display em uma bancada de testes automatizada (informalmente chamada de jiga de testes). Durante os testes, as novas informações de hardware do display são armazenadas em um banco de dados. Há casos, entretanto, em que técnicos a serviço da Hexagon realizam as trocas de hardware em campo. Nesse caso, torna-se desafiador manter os registros atualizados e isso gera discrepâncias no momento da consulta do script ao banco de dados - resultando em senhas incorretas;
- Historicamente, existem duas versões do algoritmo que gera as senhas de SSH, sendo o ponto de quebra de compatibilidade a troca do *kernel* do sistema operacional do display. Displays antigos podem ter saído de fábrica com uma senha gerada pela versão anterior. Esse problema foi parcialmente mitigado obtendo uma versão do programa de geração de senhas com o algoritmo antigo e, caso a versão original não seja capaz de determinar a senha, é possível realizar outra tentativa utilizando o programa legado, mas esse processo é manual e não foi automatizado pelo script, pois é necessário compilar o programa para a arquitetura do computador em que o script for executado antes de utilizá-lo (o que também caracteriza um problema de compatibilidade).

Portanto, a necessidade de descobrir a senha de SSH do display por um método de tentativa e erro durante a conexão é um grande desafio de usabilidade da solução existente, e muitas tentativas de acesso falham por esse fator.

3.2.3.3 Ausência de registro de sessões

Com a solução atual, não é possível controlar ou ao menos auditar quem acessa o terminal de um determinado display, e quais comandos são enviados para o mesmo. Isso representa um risco de segurança para a empresa, sendo que as máquinas em que os displays estão instalados devem manter seu status operacional durante a sessão, e o operador pode não estar ciente de que alguém está acessando o terminal do display conectado à máquina que ele está operando. Se o acesso remoto interferir de alguma forma nos dados gerados pela aplicação (por exemplo, ocasionando a interrupção de um dos serviços embarcados), a empresa pode ser culpabilizada. Há contratos com clientes corporativos em que a situação de perda de dados pode resultar em multas de valor considerável. No caso do AgrOn Acesso Remoto, um ícone foi criado para exibição na interface quando o *streaming* da tela está ativo, mas não existe um indicador similar para o *ReverseSSH*.

3.3 Soluções comerciais para acesso remoto via terminal

Foi realizada uma breve pesquisa de mercado para entender como os competidores diretos da Hexagon e demais empresas do mercado de eletrônica embarcada gerenciam o acesso remoto aos seus produtos. Os resultados são relatados abaixo.

3.3.1 Competidores diretos

- *John Deere*[2], fabricante estadunidense de máquinas e equipamentos agrícolas, também possui uma linha de displays, para os quais disponibiliza uma ferramenta chamada *Remote Display Access*. Essa ferramenta permite que técnicos possam visualizar a tela dos displays da *John Deere* e auxiliar os operadores na configuração dos equipamentos e na investigação de problemas. A documentação on-line é limitada, mas foi possível confirmar, por meio de um material de treinamento da *John Deere* direcionado a parceiros de revenda, que a ferramenta não permite o controle da tela dos displays. Não foi encontrada menção ao acesso em modo de terminal.

- *Trimble AG*[3], divisão de agricultura da empresa americana *Trimble*, não possui uma ferramenta de acesso remoto proprietária para seus displays, mas utiliza uma versão embarcada do *TeamViewer*, que é uma solução já estabelecida de acesso remoto para *desktops*. O *TeamViewer* permite tanto a visualização como o controle da tela do sistema no qual está instalado, e também é possível o acesso em modo de terminal.

- Stara[4], empresa brasileira que também fabrica máquinas agrícolas e displays, possui um serviço chamado Conecta, em que técnicos da empresa fornecem suporte remoto por meio de um canal no WhatsApp, pelo qual o operador envia as informações do equipamento a ser acessado e se comunica com o técnico alocado para a solicitação. Além de visualizar a tela do display, o técnico pode controlá-la e realizar configurações remotamente.

3.3.2 Testes com o *Shellhub*

Desenvolvido pela *O.S. Systems*, empresa brasileira fundada em 2002 que oferece soluções customizadas para sistemas embarcados, o *Shellhub*[5] é uma plataforma de acesso remoto para sistemas Linux multiarquitetura, com funcionalidades como controle de acesso, autenticação via chave e histórico de sessão. Está disponível para a maioria das distribuições Linux, e consiste em um agente, que deve ser instalado no

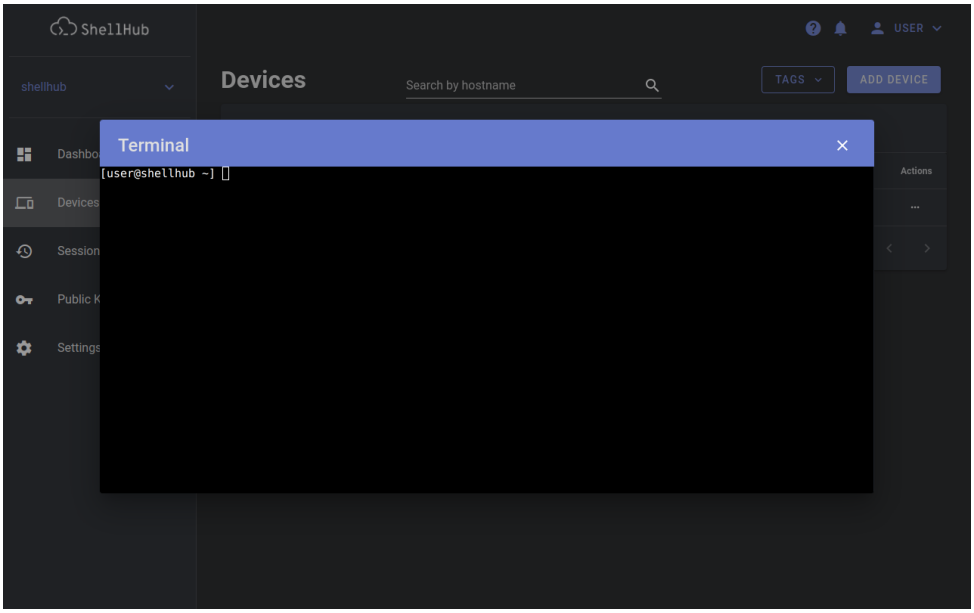
[2] https://www.deere.com.br/
[3] https://agriculture.trimble.com/
[4] https://stara.com.br/
[5] https://www.shellhub.io/

dispositivo a ser acessado remotamente, e um aplicativo web, oferecido nas modalidades *self-hosted*, instalado em servidor local do cliente, ou *managed*, hospedado em um servidor gerenciado pela própria *O.S. Systems*.

Durante a execução do projeto, uma reunião remota foi agendada com representantes da *O.S. Systems* para conhecer mais sobre a ferramenta e obter uma licença de teste de 30 dias no plano *Enterprise*, a fim de realizar testes e estabelecer uma linha de base a partir da qual o projeto desenvolvido será analisado.

A Figura 18 apresenta a tela de sessão do *Shellhub*.

Figura 18 – *Shellhub* - tela de sessão.



Fonte: Arquivo pessoal.

Para testar o *Shellhub* nos displays da Hexagon, uma *layer* disponibilizada no repositório *meta-shellhub*(https://github.com/shellhub-io/meta-shellhub) foi adicionada ao processo de compilação do *Yocto* para as arquiteturas dos displays Ti5, Ti7 e Ti10. Essa *layer* é responsável por instalar o agente embarcado do *Shellhub* nos displays, como um serviço gerenciado pelo *systemd*.

As principais funcionalidades do *Shellhub*, precificação da ferramenta e algumas considerações a respeito da usabilidade dentro do contexto dos displays da Hexagon são descritas a seguir.

3.3.2.1 Controle de acesso

Pelo painel de administrador, é possível criar usuários e realizar associações usuário/dispositivo(s). O *Shellhub* utiliza o conceito de *namespace*, que compreende um conjunto de dispositivos, e a associação também pode ser realizada entre usuá-

rio e *namespace*, de forma que um usuário ganha acesso a todos os dispositivos cadastrados no *namespace* associado ao mesmo.

3.3.2.2 Opções de autenticação

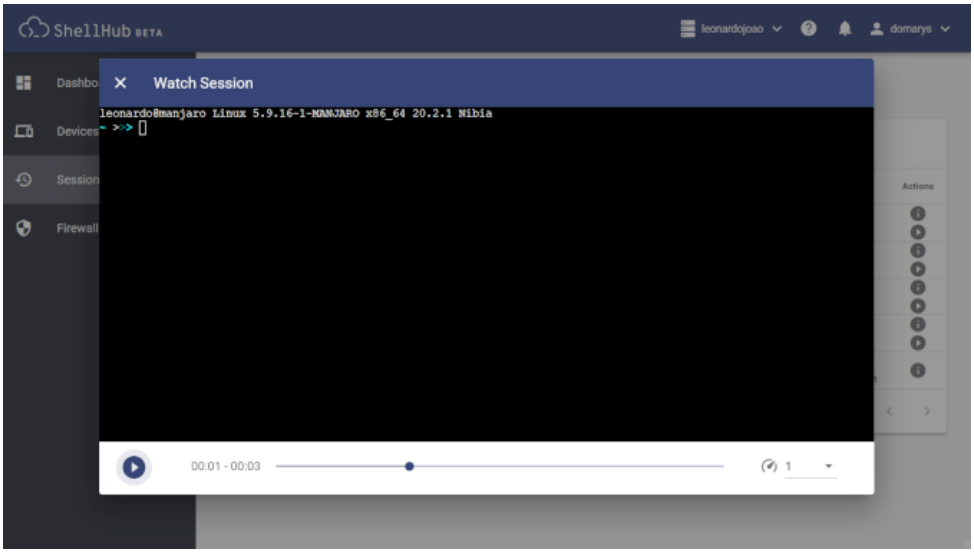
É possível realizar a autenticação nos dispositivos de duas formas: com o par usuário/senha de SSH ou com uma chave pública, cuja respectiva chave privada é registrada no painel de administrador. Cada chave pública cadastrada deve ser associada aos respectivos usuários e displays. O agente embarcado do *Shellhub* realiza o registro automático da chave pública associada a um certo dispositivo à lista de chaves de SSH permitidas.

3.3.2.3 Registro de sessões

O painel de sessões do *Shellhub* disponibiliza o histórico geral de sessões (não existe filtro por usuário ou dispositivo). Ao lado de cada registro de sessão, é possível clicar no botão *"play"* e assistir uma gravação, em um formato de *playback* de caracteres ASCII, conforme recebidos e transmitidos pelo terminal.

A Figura 19 apresenta a tela de *playback* de sessão do *Shellhub*.

Figura 19 – *Shellhub* - tela de *playback* de sessão.



Fonte: (OPENSOURCE.COM, 2021)

3.3.2.4 Precificação

Existem dois planos para o *Shellhub*: o plano *Community*, voltado para desenvolvedores individuais, e o *Enterprise*, voltado para empresas que precisam gerenciar um número considerável de dispositivos.

O plano *Community* é gratuito e é oferecido na modalidade *self-hosted*, porém alguns recursos não estão disponíveis, como o registro de sessões, criptografia do tráfego de mensagens, e o acesso ao painel de administrador, o que significa que novos usuários e *namespaces* não podem ser criados com esse plano.

Já em relação ao plano *Enterprise*, que desbloqueia todos os recursos do Shellhub, a modalidade disponível é a *managed* (serviço hospedado e gerenciado pela própria O.S. Systems), e o valor cobrado é proporcional ao numero de dispositivos cadastrados, até o teto de 400 dispositivos, a partir do qual o valor se mantém o mesmo. O preço mensal para 400 dispositivos ou mais é de cerca de 800 dólares, havendo também a possibilidade de cobrança anual por cerca de 8500 dólares, o que representa aproximadamente 700 dólares mensais.

3.3.2.5 Dados de performance

Como cenário para obtenção de dados de performance, o *Shellhub* foi instalado em um display de teste e o comando *top* foi executado no terminal de cada um dos displays pela aplicação web. Este comando produz uma lista de processos do sistema classificados por uso de memória, CPU e outros parâmetros, que é atualizada periodicamente (por padrão, a cada segundo). Os seguintes dados foram coletados:

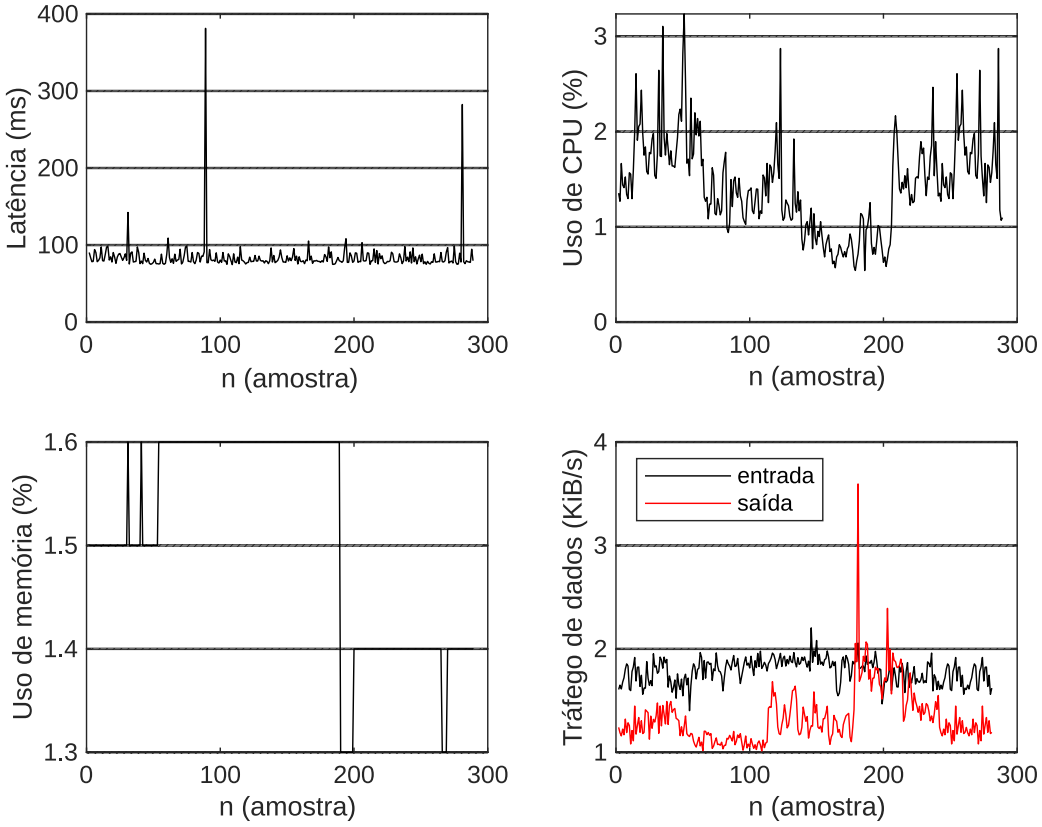
- Uso de memória do display;
- Uso de CPU do display;
- Uso de dados da rede do display;
- Latência da conexão entre o display e a aplicação web.

Para a obtenção dos dados relacionados a uso de recursos do display, um script foi elaborado para coletar as informações a partir do resultado do comando *top*, e registrá-las em um arquivo .csv para posterior análise. Já quanto aos dados de rede, o software *Wireshark* foi utilizado para capturar os pacotes de entrada e saída da aplicação.

A Figura 20 apresenta os resultados do teste realizado. Os dados foram coletados a cada 5 minutos durante um período de 24 horas.

A Tabela 2 apresenta os resultados sumarizados do teste.

Figura 20 – *Shellhub* - dados de performance da aplicação.



Fonte: Arquivo pessoal.

Tabela 2 – Resultados do teste de performance com o *Shellhub*.

Medida	Valor médio	Pico
Uso de CPU do display (%)	1.9	3.2
Uso de memória do display (%)	1.4	1.6
Tráfego de dados de entrada (KiB/s)	1.3	3.6
Tráfego de dados de saída (KiB/s)	1.7	2.2
Latência da conexão com o servidor (ms)	85	382

Fonte: Arquivo pessoal.

3.3.2.6 Considerações finais

Os testes com o *Shellhub* indicaram que a ferramenta é estável, confiável e apresenta extensa documentação, suporte eficiente e riqueza em funcionalidades. A vantagem mais expressiva de utilizar um sistema de acesso remoto gerenciado por terceiros seria isentar os times de desenvolvimento da tarefa de manutenção e suporte, que é dispendiosa e um dos principais gargalos para o planejamento do *roadmap* da Hexagon.

Os principais pontos de preocupação encontrados durante os testes são: a impossibilidade de integração com outros sistemas, como uma API para o gerenciamento das contas de usuário e permissões, já que a Hexagon tem o seu próprio banco de dados já integrado ao AgrOn Acesso Remoto; o modelo de *playback* de sessão, considerado ineficiente e sem suporte a consultas com filtro por usuários e comandos específicos; e o modelo de precificação por número de dispositivos cadastrados. A Hexagon possui um numero elevado de displays ativos que devem estar sempre disponíveis para acesso, porém a realização efetiva de sessões é esporádica e ocorre com uma parcela muito reduzida desses displays. Estima-se que cerca de cinco a dez displays sejam acessados remotamente por mês no cenário atual, o que tornaria a cobrança no nível de mais de 400 dispositivos pouco interessante para a empresa.

Por conta disso, optou-se por desenvolver um sistema proprietário que esteja integrado com os demais serviços da Hexagon, mesmo que ele eventualmente seja mais simples ou menos responsivo que o *Shellhub*, desde que os objetivos propostos sejam cumpridos (auditoria de sessões e controle de acesso). A modelagem e desenvolvimento desse sistema será descrita nos próximos capítulos.

4 Modelagem

Neste capítulo, apresenta-se a etapa de modelagem do projeto, que envolve a elaboração de diagramas para ilustrar os fluxos de informação do sistema, levantamento de requisitos funcionais e não-funcionais, definição de casos de uso e estruturas de dados. As etapas desse capítulo baseiam-se no modelo de Processo Unificado (UP) descrito em Wazlawick (2016), livro-texto da disciplina de Metodologia para Desenvolvimento de Sistemas.

4.1 Descrição conceitual do sistema

O sistema deve permitir que o usuário selecione um display ao qual ele tem permissão para acessar remotamente, e envie comandos por meio de um terminal interativo. Além disso, o usuário deve ser capaz de consultar o histórico de sessões passadas, visualizando os comandos enviados.

Com base nos objetivos delimitados para o sistema, os seguintes requisitos funcionais e não funcionais podem ser extraídos.

4.1.1 Requisitos funcionais

- O sistema deve renderizar um terminal interativo e aceitar o *input* de comandos pelo usuário.
- O sistema deve armazenar o histórico de comandos enviados aos displays.

4.1.2 Requisitos não funcionais

- O sistema deve possuir controle de acesso, isto é, apenas determinados usuários poderão acessar o modo de terminal.
- O sistema deve ser composto por uma aplicação web baseada nos serviços da AWS e implementada utilizando o *framework* Angular, e um agente embarcado responsável pela comunicação com o display.
- A autenticação no lado do display não deve estar associada à senha de SSH.
- O sistema deve utilizar os bancos de dados e estruturas já implementados para o AgrOn Acesso Remoto.
- O sistema deve interagir com o display de forma que a performance das demais aplicações embarcadas não seja afetada negativamente pela sua existência.

4.2 Escopo de projeto

Como visto no capítulo 3, o AgrOn Acesso Remoto já tem uma infraestrutura estabelecida, tanto na web como no próprio display, e está em operação no momento. Qualquer implementação aqui realizada precisa ser integrada ao sistema já existente. O controle de acesso, com tipos especiais de usuário (administrador, super usuário, entre outros) já foi implementado no AgrOn Acesso Remoto, portanto a ferramenta desenvolvida precisa apenas integrar essa funcionalidade, e o modo de terminal deve ser acessível apenas aos usuários do tipo "*special_user*", que ficam armazenados em uma tabela específica no banco de dados.

O que deve ser efetivamente desenvolvido no projeto consiste em um terminal interativo no *front-end* do sistema de acesso remoto, os componentes de *back-end* necessários para o funcionamento do terminal, e o módulo embarcado, a ser integrado ao *Wanda*.

4.3 Casos de uso

Por meio da descrição conceitual e da especificação de requisitos, é possível levantar os casos de uso do sistema e delimitar o escopo da aplicação.

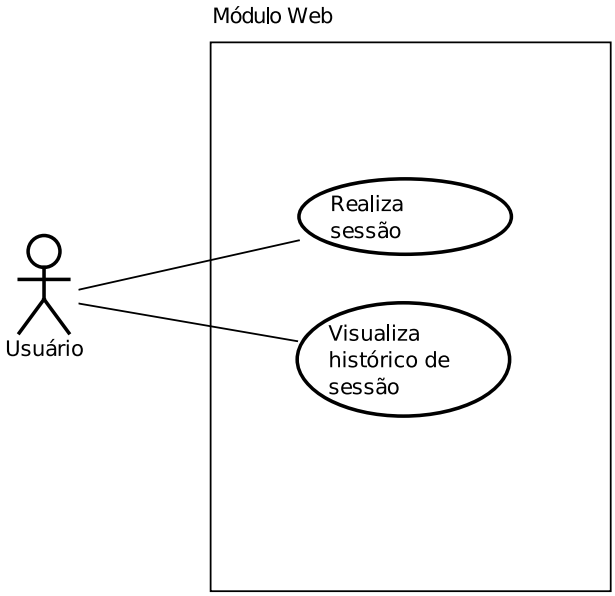
Um caso de uso é definido como uma operação indivisível realizada pelo usuário. Com essa definição, observa-se que o sistema proposto possui dois casos de uso: sessão de acesso remoto ao terminal de um display e visualização de histórico de sessões.

Passos intermediários como login, seleção de displays ou envio de comandos não são considerados casos de uso, pois não é possível executar esses passos separadamente, eles apenas existem dentro do contexto da sessão.

Quanto aos atores de sistema, o único ator identificado é o usuário, que interage com o sistema por meio da realização de sessões e visualização do histórico. O display não é considerado um ator porque não é capaz de iniciar nenhum dos dois casos de uso por conta própria.

O diagrama de casos de uso de alto nível é apresentado na Figura 21.

Figura 21 – Diagrama de casos de uso de alto nível para o sistema.



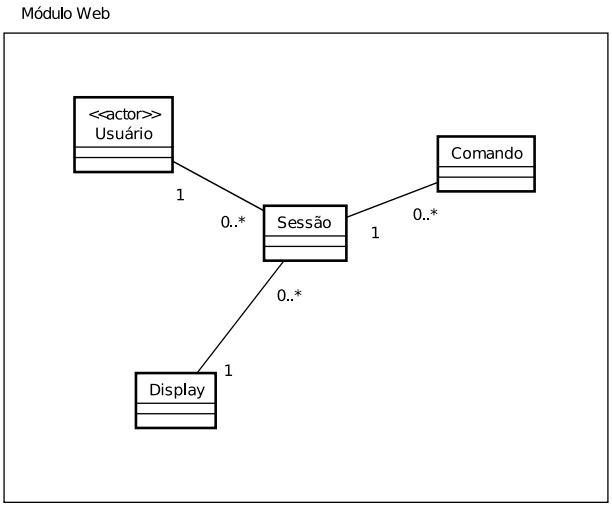
Fonte: Arquivo pessoal.

4.4 Modelo conceitual

A próxima etapa da modelagem consiste em mapear as classes necessárias para representar as estruturas de dados do sistema e o relacionamento entre elas. Os atributos e métodos associados a cada classe serão adicionados posteriormente.

A Figura 22 apresenta o modelo conceitual do sistema.

Figura 22 – Modelo conceitual do sistema.



Fonte: Arquivo pessoal.

4.5 Expansão dos casos de uso

Na próxima etapa de modelagem, é elaborada uma descrição textual do fluxo principal dos casos de uso. Analisa-se esse fluxo para a identificação de possíveis exceções. Após identificar uma possível exceção, um procedimento para corrigir o problema é descrito em um fluxo alternativo.

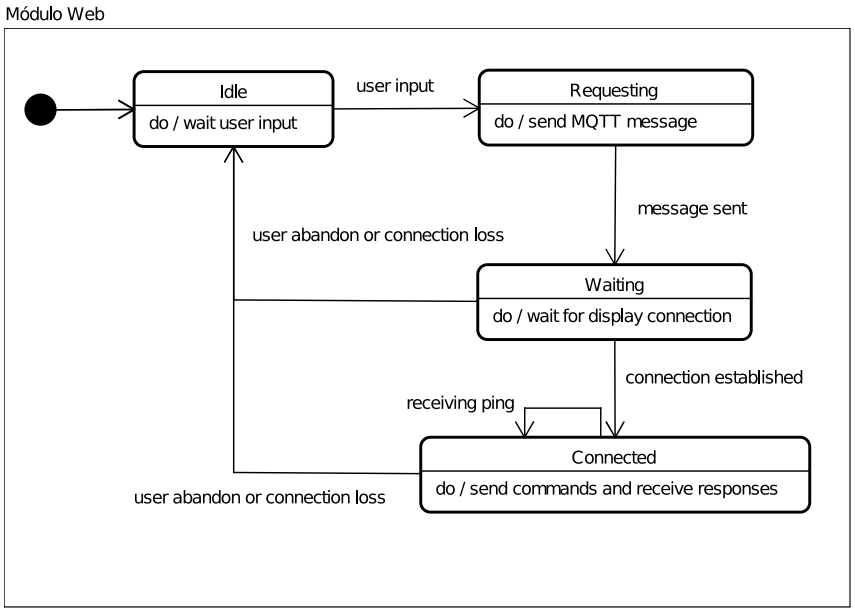
4.5.1 Caso de uso 01: realiza sessão

A descrição do caso de uso é realizada abaixo.

1. O usuário clica no botão Abrir terminal.
2. A sessão é iniciada.
3. O sistema envia uma mensagem via tópico MQTT para o display solicitando uma conexão.
4. O sistema aguarda a resposta do display.
   - 4a. **Exceção:** a conexão com o display é perdida.
     - 4a.1. A sessão é encerrada.
     - 4a.2. Retorna ao passo 1.
   - 4b. **Exceção:** o display recusa a conexão.
     - 4b.1. A sessão é encerrada.
     - 4b.2. Retorna ao passo 1.
   - 4c. **Variante:** o usuário clica no botão Finalizar Acesso.
     - 4c.1. A sessão é encerrada.
     - 4c.2. Retorna ao passo 1.
5. O terminal é exibido.
6. O usuário envia comandos para o display por meio do terminal interativo, e visualiza as respostas. O sistema registra os comandos enviados ao display.
   - 6a. **Exceção:** a conexão com o display é perdida.
     - 6a.1. A sessão é encerrada.
     - 6a.2. Retorna ao passo 1.
   - 6b. **Variante:** o usuário clica no botão Finalizar Acesso.
     - 6b.1. A sessão é encerrada.
     - 6b.2. Retorna ao passo 1.

O diagrama de máquina de estados correspondente à descrição acima é apresentado na Figura 23.

Figura 23 – Diagrama de máquina de estados para o caso de uso 01.



Fonte: Arquivo pessoal.

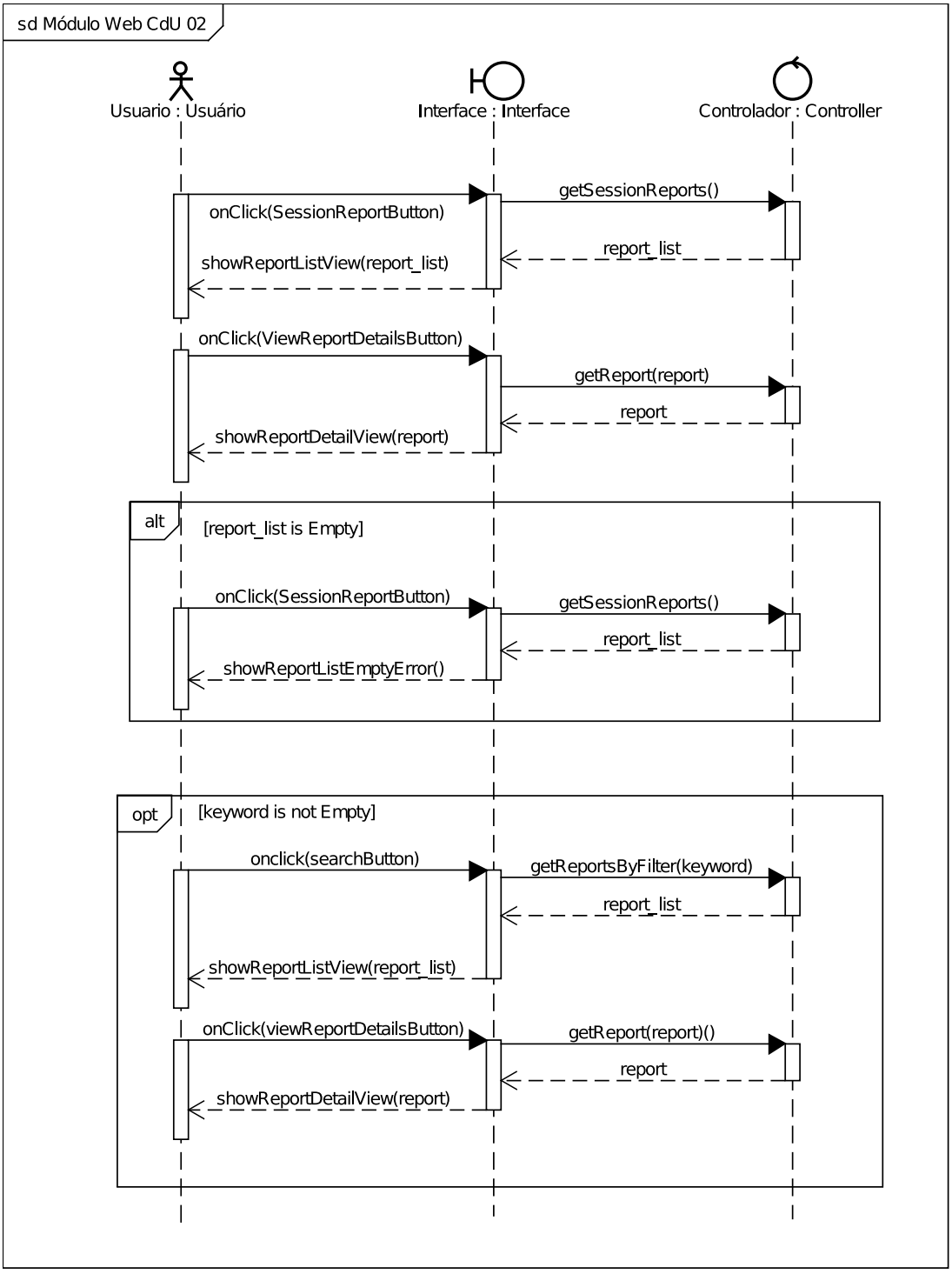
4.5.2 Caso de uso 02: visualiza histórico de sessões

A descrição do caso de uso é realizada abaixo.

1. O usuário clica no botão Histórico de Sessões.
2. O sistema carrega a página de histórico de sessões.
3. O usuário seleciona a sessão desejada.
   - 3a. **Exceção:** Não há sessões para seleção.
     - 3a.1. O sistema informa o usuário.
     - 3a.2. Retorna ao passo 2.
   - 3b. **Variante:** O usuário filtra as sessões por palavra-chave.
     - 3b.1. O sistema carrega novamente a página de histórico de sessões com o filtro aplicado.
     - 3b.2. Retorna ao passo 3.
4. O sistema exibe o histórico da sessão selecionada.

O diagrama de sequência referente ao caso de uso 02 é apresentado na Figura 24.

Figura 24 – Diagrama de sequência para o caso de uso 02.



Fonte: Arquivo pessoal.

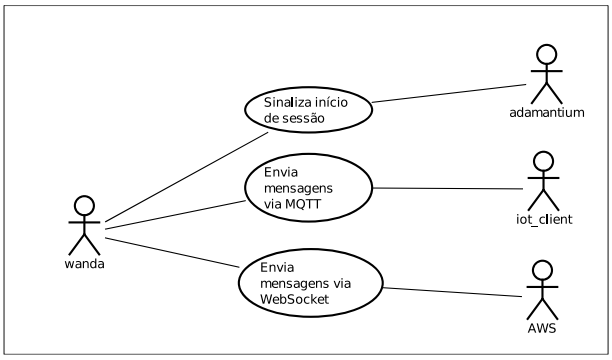
4.6 Módulo embarcado

O módulo embarcado possui a função de enviar e receber mensagens via *WebSocket* e tópicos MQTT, realizando a interface entre o display e a aplicação web. Quanto às interações de atores com o sistema, é possível identificar que o módulo embarcado tem interações entre os seguintes atores:

- Serviço *Wanda*, que troca mensagens com outros serviços e se conecta à aplicação web via *WebSocket*.
- Serviço *iot_client*, que recebe e envia mensagens para o *Wanda*.
- Serviço *adamantium*, que recebe um sinal do *Wanda* de que o acesso remoto foi estabelecido para exibir um ícone na tela do display.
- *AWS*, representando a infraestrutura de nuvem.

O diagrama de casos de uso de alto nível para o módulo embarcado é apresentado na Figura 25.

Figura 25 – Diagrama de casos de uso de alto nível para o módulo embarcado.



Fonte: Arquivo pessoal.

O próximo capítulo trata de detalhes de implementação do sistema, como serviços da AWS e conceitos de controle de processos de terminal.

5 Desenvolvimento

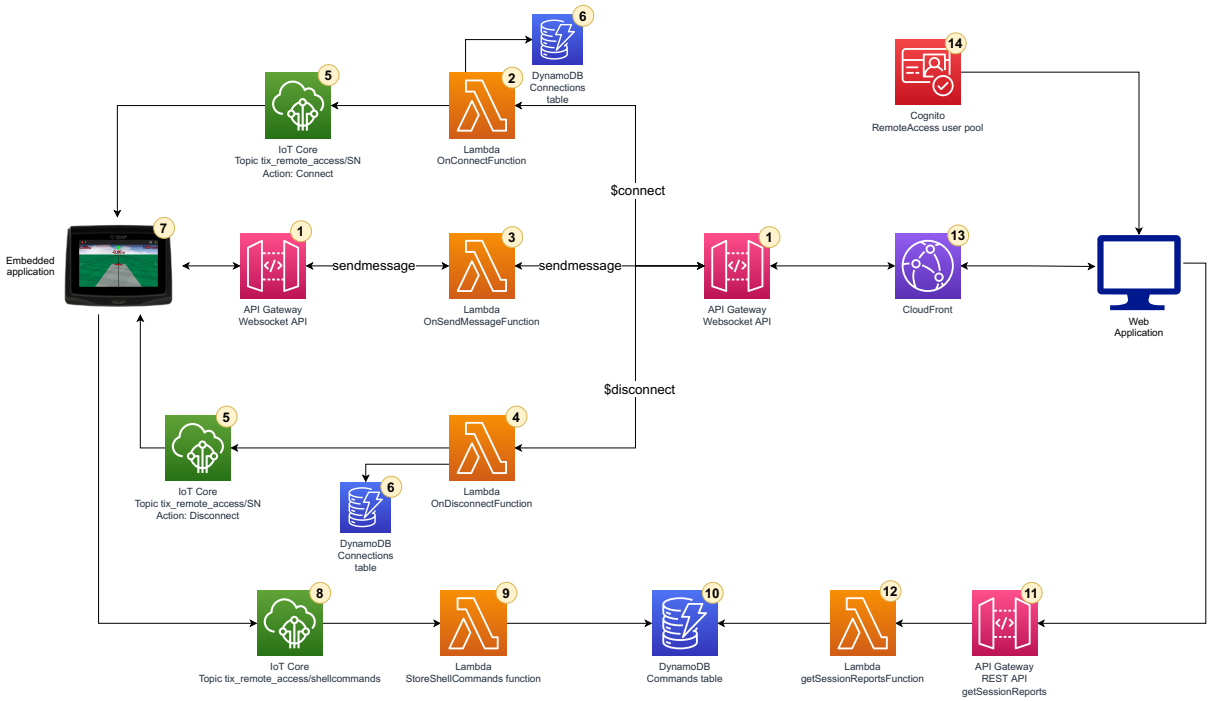
Neste capítulo serão apresentados os conceitos, tecnologias e ferramentas que foram utilizados durante a etapa de implementação do projeto.

Nas seções 5.1 e 5.2, apresenta-se uma visão geral da arquitetura implementada na AWS, que é complementada por uma introdução aos serviços da AWS, realizada na seção 5.3. A seção 5.4 contém uma introdução ao *framework* de desenvolvimento *Angular*, utilizado no *front-end*, e a seção 5.5 detalha alguns conceitos relevantes para o controle de entrada e saída em processos, que foram utilizados para implementação do acesso remoto via terminal.

5.1 Diagrama de arquitetura

A Figura 26 apresenta o diagrama de arquitetura do sistema, com os serviços da AWS utilizados na infraestrutura de nuvem.

Figura 26 – Diagrama de arquitetura do sistema.



Fonte: Arquivo pessoal.

Os serviços utilizados são enumerados abaixo.

1. API *WebSocket*: conjunto de rotas *$connect*, *$disconnect* e *sendmessage*. As rotas *$connect* e *$disconnect* já estavam implementadas no sistema original, mas as *lambdas* associadas às mesmas foram alteradas para comportar a nova

funcionalidade. Já a rota *sendmessage* foi adicionada e é o canal principal de comunicação entre o terminal web e o display.

2. *Lambda OnConnectFunction*: essa função foi alterada para criar um novo registro na tabela de conexões criada para a funcionalidade no *DynamoDB* (item 6). Essa tabela de conexões é separada pois é utilizada para o controle de sessões de terminal ativas, no caso de mais de um usuário acessando o mesmo display. Quando o display envia o resultado de um comando, a *lambda* associada à rota *sendmessage* consulta a tabela criada para saber quais conexões ativas estão associadas ao display, e fazer o devido encaminhamento das mensagens.

3. *Lambda OnSendMessageFunction*: é a função de redirecionamento de mensagens do sistema. Durante a sessão, todas mensagens enviadas passam por essa rota, e o corpo da mensagem, que está no formato JSON, possui um remetente e um destinatário, que são processados pela *lambda* para que cada comando seja direcionado para o *WebSocket* correto.

4. *Lambda OnDisconnectFunction*: complementar ao item 2, essa função é responsável por remover as conexões da tabela no DynamoDB após o final de uma sessão.

5. Tópico *tix_remote_access/SN*: assim como no AgrOn Acesso Remoto original, esse tópico permite a comunicação do sistema web com o display por meio do serviço *iot_client*, solicitando o início ou final de uma sessão. Com a implementação do projeto, uma mudança adicionada é que, quando o usuário solicita o início de uma sessão no modo de terminal, uma nova mensagem é enviada para esse tópico com o *link* do *WebSocket* ao qual o display deve se conectar. Essa mensagem é escutada pelo *iot_client* e redirecionada pelo *wanda*, que efetua a conexão.

6. Tabela *AgrOnRemoteAccessShellConnections*: tabela utilizada para o registro de conexões ativas durante uma sessão, consultada pelas *lambdas* dos itens 2, 3 e 4. A estrutura dessa tabela será apresentada na próxima seção.

7. Módulo embarcado: aplicação executada a partir do serviço *Wanda* que realiza a conexão via *WebSocket*, a execução de comandos e o envio de respostas durante uma sessão.

8-9. Registro de comandos: quando um comando é executado pelo módulo embarcado, uma mensagem MQTT é enviada para o tópico *tix_remote_access /shellcommands* registrando sua execução, como parte da implementação da funcionalidade de histórico de sessões. Os comandos são armazenados em uma tabela no *DynamoDB* chamada *AgronRemoteAccessShellCommands*.

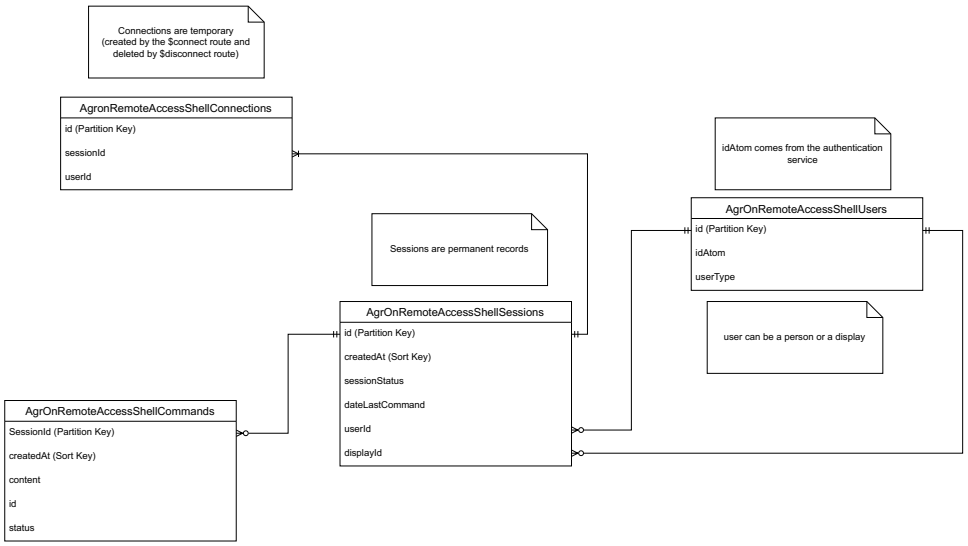
11-12. Consulta ao histórico de sessões: quando o usuário solicita a visualização do histórico de uma sessão, uma mensagem é enviada pelo *endpoint* criado na API REST do AgrOn Acesso Remoto para a obtenção da lista de sessões.

13 *CloudFront*: responsável pelo provisionamento dos recursos estáticos e dinâmicos da página web do AgrOn Acesso Remoto, como o módulo de terminal interativo.

5.2 Modelo lógico dos bancos de dados

A Figura 27 apresenta o modelo lógico do banco de dados do projeto no *DynamoDB*, com as tabelas necessárias para implementar o modelo conceitual da Figura 22.

Figura 27 – Modelo lógico do banco de dados do sistema.



Fonte: Arquivo pessoal.

A plataforma AWS e detalhes sobre os serviços utilizados para construção da arquitetura são apresentados a seguir.

5.3 AWS (*Amazon Web Services*)

Esta seção inclui informações sobre a plataforma AWS, utilizada pela Hexagon em suas soluções baseadas em processamento na nuvem, incluindo o AgrOn Acesso Remoto.

Serão apresentados os principais serviços da plataforma, e como eles foram utilizados no contexto do projeto.

5.3.1 Introdução à AWS

A AWS é uma plataforma da *Amazon.com, Inc* para soluções de computação na nuvem. Atualmente, oferece cerca de 200 serviços para finalidades como armazenamento de arquivos, capacidade computacional, bancos de dados, hospedagem de sites, autenticação de usuários e integração de sistemas via API (*Application Programming Interface*) (AMAZON WEB SERVICES, c2022f). Todas as funcionalidades que uma aplicação web pode demandar são contempladas pela AWS, apesar de ela não ser a única plataforma do tipo, havendo competidores como a *Azure*, da Microsoft, e a *Google Cloud Platform*.

Soluções implementadas com a AWS são inerentemente escaláveis e seguras, e os desenvolvedores não precisam se preocupar com aspectos como a disponibilidade dos sistemas, sendo a plataforma responsável por garantir essa e outras propriedades de interesse no âmbito das aplicações em nuvem.

Quanto à precificação dos serviços AWS, o modelo utilizado é o de cobrança proporcional à demanda e complexidade da tecnologia utilizada. Ou seja, serviços de armazenamento se tornam mais caros de acordo com o aumento dos dados armazenados, APIs são cobradas conforme o número de requisições recebidas e processadas, e serviços de computação na nuvem são cobrados de acordo com o tempo de uso e o custo computacional.

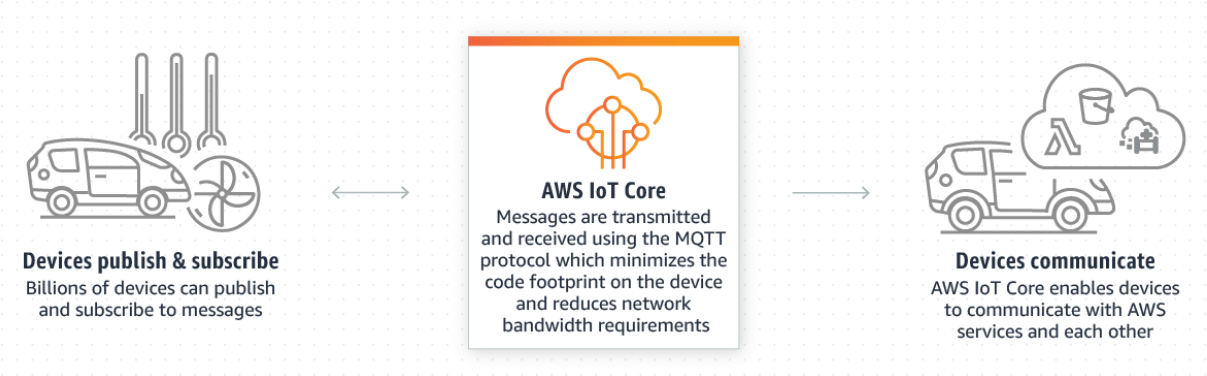
Os serviços da AWS relevantes para o projeto são descritos abaixo.

5.3.2 *IoT Core*

IoT Core é o serviço da AWS destinado à transmissão de mensagens a dispositivos e aplicações de IoT, categoria na qual os displays da Hexagon se encaixam. Aceita o protocolo MQTT *(Message Queuing Telemetry Transport)*, considerado o padrão da indústria para aplicações no modelo pub/sub (publicador/subscritor), e também os protocolos LoRaWAN (para dispositivos de radiofrequência) e HTTPS (para comunicação unidirecional, menos comum no contexto de IoT).

Os casos de uso do *IoT Core* incluem o monitoramento de operações industriais remotas e integração de assistentes virtuais com dispositivos de automação residencial (AMAZON WEB SERVICES, c2022b). Nas soluções da Hexagon, o *IoT Core* é um componente essencial, utilizado principalmente para a transmissão de mensagens de monitoramento do display, as quais contém dados de telemetria que são processados por outros serviços e alimentam as aplicações web, como a Sala de Controle. A Figura 28 apresenta um exemplo de uso do *IoT Core*, extraído da documentação da AWS.

Figura 28 – Caso de uso do *IoT Core*.



Fonte: (AMAZON WEB SERVICES, c2022b)

5.3.3 *Lambda*

Lambda é um serviço da AWS voltado para computação em nuvem sem a necessidade do provisionamento de servidores, também conhecida como computação *serverless*. Aplicações que utilizam *Lambda* consistem em um grupo de funções que são disparadas independentemente por eventos originados de outros serviços da AWS. Essas funções são comumente chamadas de funções *Lambda*, ou simplesmente *lambdas*.

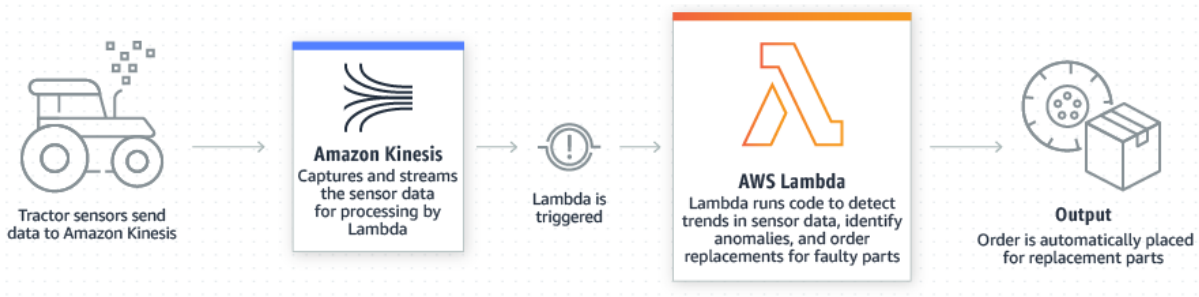
A alocação de recursos para o processamento das funções é dinâmica, garantindo a escalabilidade, e o serviço elimina preocupações com a disponibilidade de servidores ou *clusters* em rede. Trata-se de um elemento de interligação entre os demais serviços da AWS, não sendo utilizado de forma isolada, mas sim recebendo eventos de um serviço, realizando o processamento dos dados associados a esse evento, e encaminhando os resultados para outro (ou para o mesmo) serviço. O principal caso de uso do *Lambda* é, portanto, a criação de aplicações orientadas a eventos, e esse também é o caso de uso principal no contexto das soluções da Hexagon, nos quais funções *Lambda* são parte da integração do *front-end* e do *back-end* das aplicações web, principalmente na implementação dos *endpoints* das APIs, e no processamento das mensagens de monitoramento recebidas via *IoT Core*.

A Figura 29 apresenta um caso de uso para uma função *Lambda* no contexto da telemetria de tratores agrícolas, que é muito similar ao encontrado nos produtos da Hexagon.

5.3.4 *RDS* (Relational Database Service)

RDS é o serviço da AWS para hospedagem e gerenciamento de bancos de dados relacionais. Possui suporte aos seguintes SGBDs (sistemas de gerenciamento de bancos de dados): *Amazon Aurora* (SGBD próprio da AWS), *MySQL*, *MariaDB*,

Figura 29 – Caso de uso de uma função *Lambda*.



Fonte: (AMAZON WEB SERVICES, c2022c)

PostgreSQL, *Oracle* e *SQL Server*. O RDS é baseado na alocação de servidores para os bancos de dados, chamados de instâncias, que podem ter diferentes especificações dependendo da necessidade do usuário, e o custo do serviço varia de acordo com o armazenamento e processamento alocados para as instâncias. O RDS também automatiza tarefas de gerenciamento como provisionamento, aplicação de patches, backup, recuperação, detecção de falhas e reparo (AMAZON WEB SERVICES, c2022g).

A Hexagon possui diversas instâncias de bancos de dados relacionais gerenciadas pelo RDS, incluindo o banco de dados do AgrOn Acesso Remoto, e a maioria delas utiliza o *PostgreSQL* como SGBD.

No projeto, não foi necessário alterar nenhuma tabela do banco de dados do AgrOn Acesso Remoto, pois elas estão associadas aos dados de cadastro de usuários e outras funcionalidades já existentes na aplicação, que foram apenas cooptadas e integradas à ferramenta desenvolvida.

5.3.5 *DynamoDB*

Em contraste com o RDS, *DynamoDB* é um gerenciador de bancos de dados não-relacionais, conhecidos genericamente como *NoSQL*[3], nos modelos de dados chave-valor e documento. Os bancos de dados *NoSQL* são empregados principalmente no armazenamento de dados heterogêneos, ou seja, que não possuem uma estrutura fixa, ou até mesmo que são seguem nenhuma estrutura (*schemaless*). Também são adotados para o armazenamento de grandes volumes de dados, apresentando ganhos de escalabilidade e elasticidade em relação aos bancos de dados relacionais (SADALAGE; FOWLER, 2013).

No contexto de computação na nuvem e aplicações web, utilizar bancos de dados não-relacionais apresenta uma vantagem pelo tempo de resposta extremamente

[3] A literatura comumente se refere a NoSQL como uma abreviação de *Not Only SQL*, mas essa definição é contestada por Sadalage e Fowler (2013), e não existe um consenso sobre o significado desse termo, que se popularizou após um encontro de desenvolvedores em São Francisco, Califórnia em 2009.

rápido das consultas realizadas. O *DynamoDB* garante uma performance abaixo de 10 milissegundos em qualquer escala (AMAZON WEB SERVICES, c2022d). A Hexagon utiliza tabelas no *DynamoDB* para armazenar informações dos displays que são consultadas por outros serviços, no serviço de chat da Sala de Controle, e no AgrOn Acesso Remoto, para armazenar os dados de conexão. A estrutura dos banco de dados relacionais e não-relacionais do AgrOn Acesso Remoto será apresentada com mais detalhes no próximo capítulo.

5.3.6 *API Gateway*

O serviço *API Gateway* é o gerenciador de APIs da AWS, permitindo a criação de APIs do tipo REST *(Representational State Transfer)*, padrão para requisições em páginas web, ou APIs Websocket, que habilitam comunicação bidirecional e assíncrona em tempo real entre aplicativos. O *API Gateway* administra o recebimento e processamento de até centenas de milhares de chamadas de API simultâneas, com suporte a gerenciamento de tráfego, controle de acesso, monitoramento e gerenciamento de versões (AMAZON WEB SERVICES, c2022a).

As APIs gerenciadas pelo *API Gateway* são comumente utilizadas em conjunto com funções *Lambda* nas aplicações, sendo o *API Gateway* responsável por instanciar os *endpoints* das APIs, e as *lambdas* responsáveis pelo processamento das requisições.

Nas soluções da Hexagon, o *API Gateway* é empregado na administração de todas as APIs das páginas web, incluindo a Sala de Controle e o AgrOn Acesso Remoto, tema deste projeto. A seguir, são realizadas algumas definições pertinentes a APIs do padrão *REST* e APIs *WebSocket*, que representam paradigmas distintos e serão essenciais para o entendimento dos desenvolvimentos propostos nos próximos capítulos.

5.3.6.1 O padrão REST

Arquitetura prevalente no design de APIs para aplicações web, REST tem o objetivo de simplificar e padronizar a comunicação entre serviços, compreendendo um conjunto de regras para a execução de operações sobre recursos da Web. APIs que seguem o padrão REST são chamadas de APIs *RESTful*. O protocolo utilizado para as requisições é o HTTP, mas uma API que usa o protocolo HTTP não é necessariamente *RESTful*, devendo aderir aos princípios de design correspondentes (RICHARDSON; RUBY, 2008).

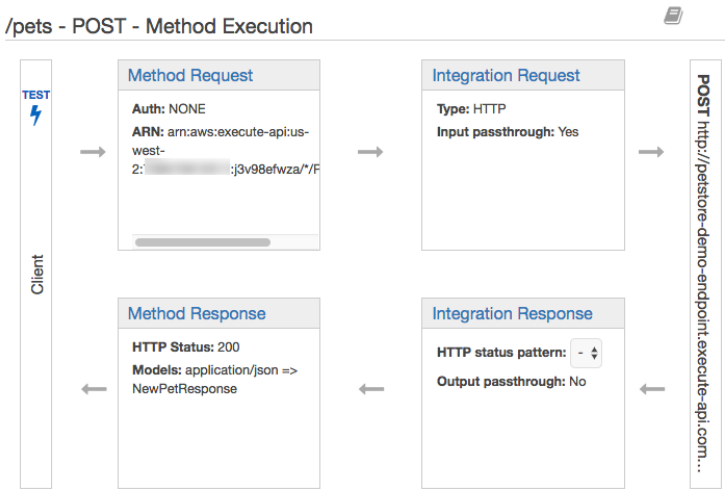
As seguintes propriedades são válidas para esse padrão:

- Modelo cliente-servidor, com separação estrita de papéis entre os atores;

- Recursos envolvidos nas requisições são unicamente identificados e representados de maneira uniforme nas respostas do servidor, no que é conhecido como URI *(Uniform Resource Identifier)*;
- As requisições do cliente para o servidor devem conter toda a informação necessária para o processamento das mesmas, não podendo depender do contexto anterior do servidor, ou seja, não há armazenamento de estados (*stateless operation*).

A Figura 30 apresenta um exemplo de um *endpoint* em uma API *RESTful*, retirado da documentação da AWS.

Figura 30 – Exemplo de *endpoint* de uma API *RESTful* no *API Gateway*.



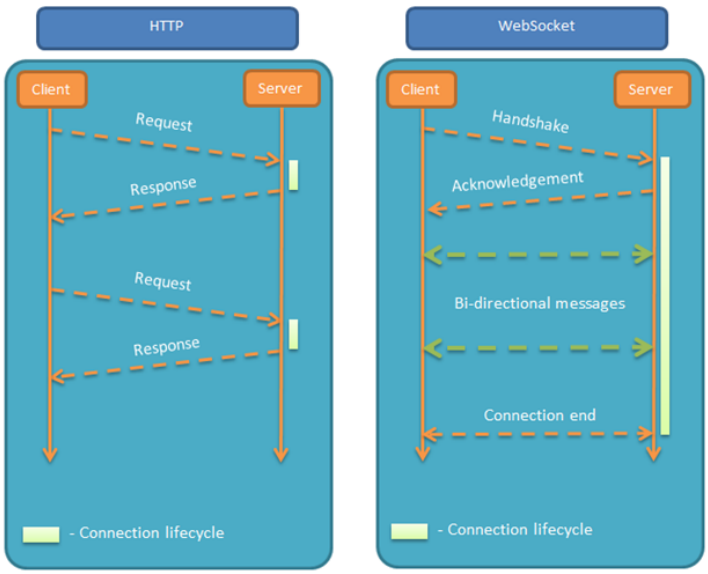
Fonte: (AMAZON WEB SERVICES, c2022i)

5.3.6.2 APIs *WebSocket*

WebSocket é um protocolo que permite a comunicação bidirecional (full-duplex) entre atores de um sistema, em contraste com o protocolo HTTP. Por isso, serviços baseados em *WebSockets* têm se popularizado desde a padronização do protocolo RFC 6455 em 2011 (MELNIKOV; FETTE, 2011), sendo encontrados principalmente no contexto de aplicações de tempo real, como mensageiros instantâneos ou *streaming* de vídeo. O *WebSocket* apresenta um contraponto ao REST por ser *stateful* e não respeitar o modelo requisição-resposta. A Figura 31 apresenta uma comparação entre os protocolos HTTP e *WebSocket*.

Na AWS, uma API *WebSocket* é definida como um conjunto de rotas configuradas de forma que mensagens recebidas pelo cliente ou pelo servidor podem ser direcionadas para diferentes integrações de *back-end* dependendo da rota selecionada. As rotas padrão são denominadas *$connect*, *$disconnect* e *$default*, sendo as

Figura 31 – Diferenças entre os protocolos HTTP e *WebSocket*.



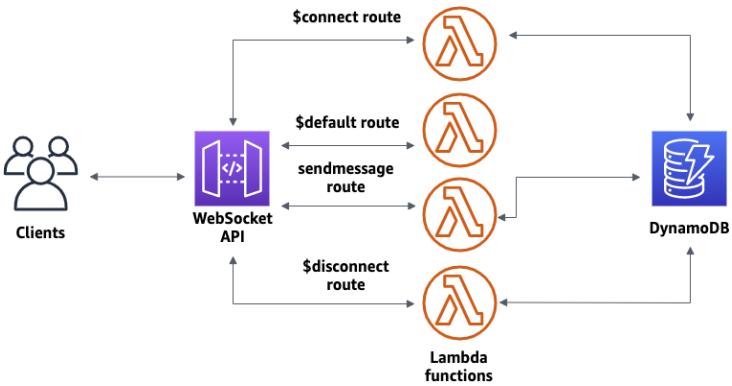
Fonte: (XORIANT, c2022)

duas primeiras responsáveis pelo controle da conexão via *WebSocket*, e a última um caminho para mensagens que não possuem uma rota específica. Novas rotas podem ser adicionadas conforme a necessidade.

As mensagens recebidas pelas rotas de uma API *WebSocket* na AWS devem estar no formato JSON (*JavaScript Object Notation*), padrão amplamente adotado para representar dados estruturados em aplicativos web.

A Figura 32 apresenta um exemplo de uso de uma API *WebSocket* para a implementação de um serviço de chat.

Figura 32 – Exemplo de aplicação da API *WebSocket*.



Fonte: (AMAZON WEB SERVICES, c2022h)

5.3.7 *CloudFront*

CloudFront é um serviço de CDN (*Content Delivery Network*, rede de entrega de conteúdo) da AWS, utilizado primariamente para o provisionamento de recursos estáticos e dinâmicos em páginas web.

A Hexagon utiliza o *CloudFront* para hospedar os recursos de suas páginas web, como a Sala de Controle e o AgrOn Acesso Remoto.

5.3.8 *Cognito*

Cognito é o gerenciador de controle de acesso para aplicações web da AWS. O armazenamento de identidades do *Cognito* pode ser dimensionado para milhões de usuários, oferecendo suporte à federação de identidades sociais e corporativas (AMAZON WEB SERVICES, c2022e) e recursos de segurança como MFA *(multi-factor authentication)*. O *Cognito* é utilizado para gerenciar o acesso de clientes da Hexagon aos serviços *AgrOn* por meio da criação de usuários.

A próxima seção traz detalhes sobre o *framework* Angular, utilizado para a construção dos componentes de *front-end* do projeto.

5.4 Angular

Angular é um *framework* de desenvolvimento para SPAs (*Single Page Applications*, aplicações de página única). SPAs são aplicações web que contêm apenas uma página HTML e atualizam essa página dinamicamente (RODRIGUES, 2016). O *Angular* é um dos três grandes *frameworks* de *front-end*, junto com o *React* e o *Vue*.

Uma aplicação desenvolvida em *Angular* é composta por componentes, módulos, e serviços. Esses conceitos são detalhados a seguir.

5.4.1 Componentes

O componente é o bloco básico de construção em *Angular*, e é equivalente a uma classe escrita na linguagem *TypeScript*, onde são definidas estruturas de dados e fluxos de aplicação. Componentes são sempre associados com telas (*views*).

5.4.2 Módulos

Módulos são coleções de componentes. Um módulo declara o contexto de compilação para um conjunto de componentes que executam ou implementam uma função específica.

5.4.3 Serviços

O conceito de serviço é utilizado para estruturas de dados ou fluxos em *TypeScript* que deseja-se reutilizar em outras partes da aplicação, porém não estão associados a nenhuma tela específica, por isso não podem ser definidos como componentes. Exemplos de funcionalidades que podem ser definidas como serviços são operações CRUD e métodos de autenticação.

5.5 Tecnologias para controle de terminal em serviços de acesso remoto

Historicamente, a palavra “terminal” se refere a um dispositivo físico que permite ao usuário se conectar a um computador central e executar comandos ou visualizar dados. Nas décadas de 60 e 70, computadores eram sinônimo de *mainframes*, ou seja, grandes unidades de processamento que não eram projetados para interação com o usuário, mas sim para a realização de tarefas complexas de processamento de dados. Portanto, as funcionalidades de comunicação com o ambiente externo não eram consideradas parte do computador, mas sim executadas por dispositivos auxiliares como os terminais.

Quanto aos tipos de terminais, é possível citar os terminais “burros” *(dumb terminals)*, que apenas implementam um canal de comunicação entre usuário e computador, e não realizam qualquer tipo de processamento; e os terminais “inteligentes” *(smart terminals)*, que possuem um microprocessador integrado e também são capazes de armazenar dados. A Figura 33 apresenta o VT100, um terminal popular durante a década de 80.

Naturalmente, com o advento dos computadores pessoais com interface gráfica, os terminais físicos deixaram de ser necessários; no entanto, a sua existência criou um paradigma para a implementação de programas de computador chamados emuladores de terminal *(terminal emulators)*, que provêm as funcionalidades dos terminais físicos de forma virtual.

Para emular o funcionamento de um terminal, os programas utilizam uma abstração chamada pseudo-terminal ou *pty*, que é descrita abaixo.

5.5.1 Pseudo-terminal

Um pseudo-terminal é par de arquivos *master/slave* que são interpretados como dispositivos físicos pelo sistema operacional, fornecem um tipo especial de canal de comunicação para processos. O dispositivo escravo se comporta como um terminal físico, como o VT100, lendo e escrevendo arquivos de texto e sequências especiais de caracteres ASCII. O mestre age como o próprio usuário, ou seja, tudo que for escrito no arquivo mestre será interpretado como o *input* de um usuário em um terminal físico.

Figura 33 – Terminal VT100.

Fonte: (OLDCOMPUTR.COM, 2015)

O conceito de *pty* possibilita a execução de processos como emuladores de terminal em sistemas operacionais (como o *xterm* ou o *gnome-terminal*) e interfaces de acesso remoto como o próprio *OpenSSH*. O que esses processos fazem efetivamente é solicitar ao sistema operacional a alocação de um pseudo-terminal que eles possam ler e controlar. Múltiplos pseudo-terminais são gerenciados por um dispositivo virtual multiplexador *(/dev/ptmx*.

5.5.2 Fluxos padrão de entrada e saída em processos Linux: *stdin*, *stdout* e *stderr*

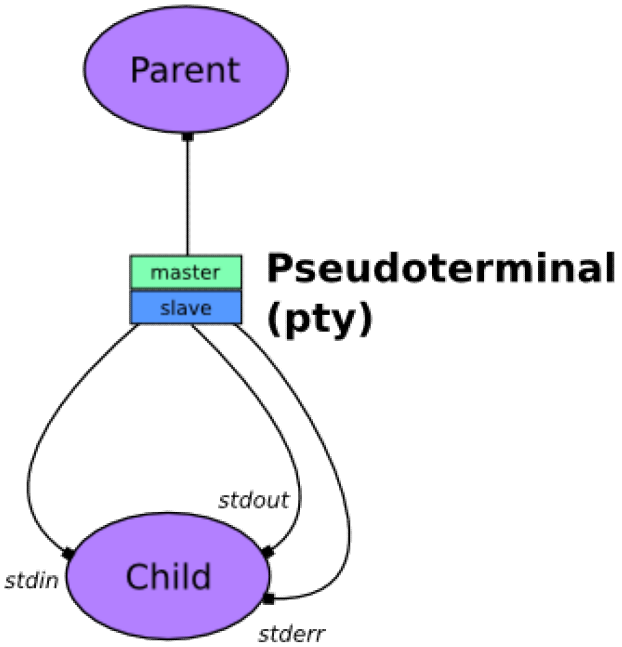
Uma forma padrão de controlar o acesso de leitura/escrita em processos de terminal é a criação de fluxos de dados (*data streams*). Esses fluxos são interpretados como arquivos pelo sistema operacional, de forma similar aos pseudo-terminais, e cada processo tem seus próprios fluxos alocados pelo sistema. Esses fluxos podem ser redirecionados para outros arquivos, ou seja, a saída de um processo pode ser escrita em um arquivo de texto específico, caso seja necessário.

O fluxo padrão de entrada de dados no Linux é chamado de *stdin*. O fluxo de saída é chamado de *stdout*, e existe também um fluxo de saída separado para mensagens de erro, chamado *stderr*. A vantagem de ter mensagens de erro em um fluxo dedicado é poder redirecionar a saída de um comando *(stdout)* para um arquivo e ainda ver qualquer mensagem de erro *(stderr)* na janela do terminal. Assim, o usuário impede que as mensagens de erro poluam o arquivo para o qual o *stdout* foi redirecionado.

No projeto, foi utilizada uma biblioteca em *Python* chamada *ptyprocess* para instanciar um pseudo-terminal a partir do qual o módulo embarcado pode executar

comandos, escrevendo no fluxo *stdin*, e lendo os resultados a partir dos fluxos *stdout* e *stderr*. A Figura 34 apresenta um esquema de como ocorre a interação do processo pai (o módulo embarcado) com o pseudo-terminal criado e seus fluxos, adaptada da documentação da biblioteca.

Figura 34 – Interação entre processo e pseudo-terminal.



Fonte: Adaptado de (KLUYVER, 2014)

.

6 Resultados

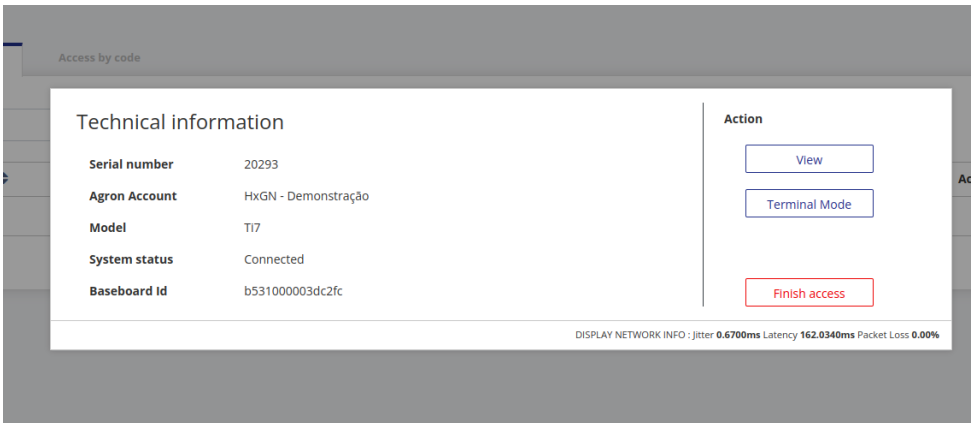
Este capítulo apresenta os resultados de implementação do projeto. A seção 6.1 apresenta as telas desenvolvidas para a aplicação web. A seção 6.2 inclui detalhes de implementação do módulo embarcado e os resultados dos testes de performance realizados, em comparação com com os resultados encontrados para o *Shellhub*. Por fim, a seção 6.3 apresenta uma análise dos resultados obtidos em termos do cumprimento dos requisitos funcionais e não-funcionais do sistema, e dos objetivos de projeto.

6.1 Telas da aplicação

Um componente de terminal interativo, baseado na biblioteca *XTerm.JS*[1] foi incorporado ao front-end do AgrOn Acesso Remoto.

A Figura 35 apresenta a tela do painel de informações do display, onde um botão para acesso ao terminal foi implementado.

Figura 35 – Painel de informações com botão de acesso ao modo de terminal.

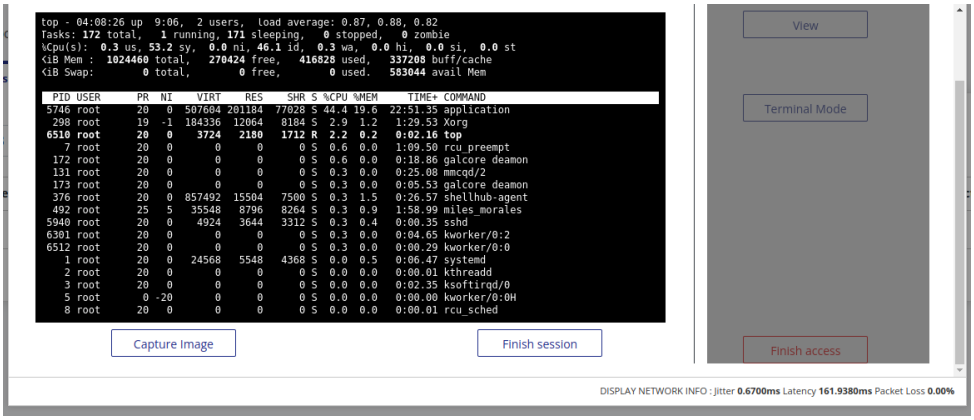


Fonte: Arquivo pessoal.

A Figura 36 apresenta a tela principal de sessão, com o componente de terminal. Finalmente, a Figura 37 apresenta a tela de histórico de sessões.

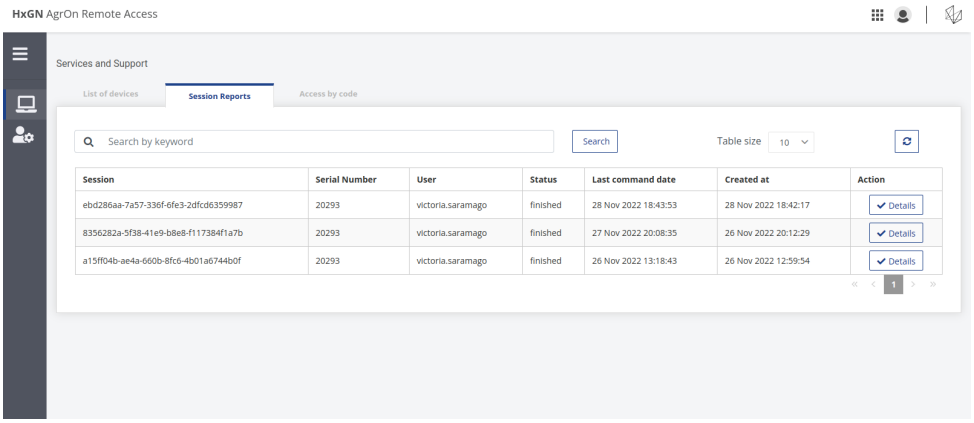
[1] https://xtermjs.org/

Figura 36 – Tela de sessão de acesso remoto.



Fonte: Arquivo pessoal.

Figura 37 – Tela de consulta de histórico de sessões.



Fonte: Arquivo pessoal.

6.2 Testes de performance

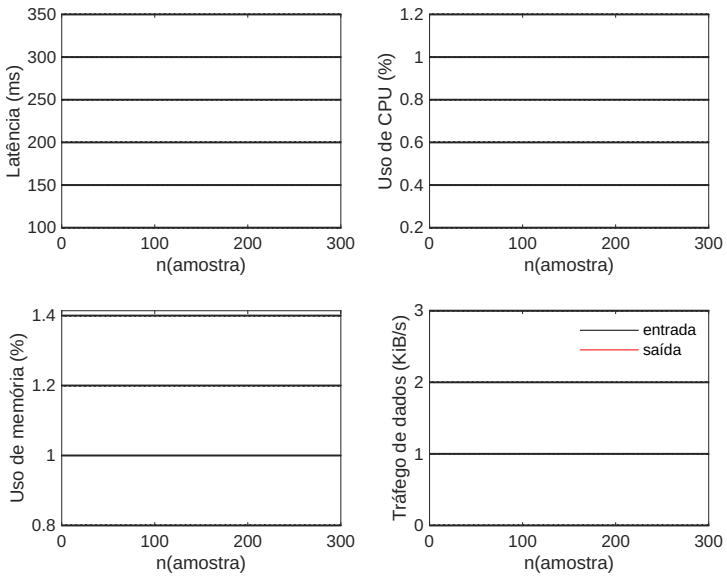
O módulo embarcado foi implementado em Python com uma biblioteca para emulação de *threads* (a linguagem não possui suporte a concorrência nativamente). Também foi utilizada uma biblioteca para implementação de funções assíncronas usando o paradigma *async/await*, recomendado para comunicação via *WebSocket*. Foram registrados os métodos *ShellSessionConnect* e *ShellSessionDisconnect* no barramento DBus, invocados por meio de mensagens recebidas e processadas pelo serviço *iot_client*, e o sinal *SignalShellSession*, consumido pelo serviço *adamantium* para instanciar um ícone na tela do display quando a sessão de acesso remoto está em andamento.

As três *threads* implementadas são descritas abaixo:

- *Thread* principal, responsável por verificar o status de conexão, aguardar o início de uma nova sessão e instanciar um pseudo-terminal para execução de comandos. Essa *thread* também é responsável por enviar mensagens MQTT registrando os comandos executados com sucesso no display;
- *Thread* de leitura e processamento de mensagens recebidas via WebSocket. As mensagens recebidas representam os comandos de terminal (*stdin*).
- *Thread* de captura da saída (*stdout* e *stderr*) do pseudo-terminal implementado, e envio de mensagens via *WebSocket*. As mensagens enviadas representam o retorno dos comandos após o processamento, em blocos de até 1024 bytes.

Dados de performance da aplicação foram coletados de maneira similar ao teste realizado com o *Shellhub* na seção 3.3, e os resultados são apresentados na Figura 38.

Figura 38 – Resultados dos testes de performance do sistema.



Fonte: Arquivo pessoal.

A Tabela 3 apresenta uma comparação entre os resultados.

Tabela 3 – Comparação de dados de performance entre o *Shellhub* e o sistema projetado.

	Shellhub		**Projeto**	
Medida	**Valor médio**	**Pico**	**Valor médio**	**Pico**
Uso de CPU do display (%)	1.9	3.2	0.5	1.1
Uso de memória do display (%)	1.4	1.6	1.1	1.4
Tráfego de dados de entrada (KiB/s)	1.3	3.6	0.5	1.0
Tráfego de dados de saída (KiB/s)	1.7	2.2	1.5	2.6
Latência da conexão com o servidor (ms)	85	382	155	337

Fonte: Arquivo pessoal.

6.3 Análise dos resultados

Os resultados encontrados podem ser analisados com base nos requisitos funcionais e não-funcionais delimitados no capítulo 4, nas seções 4.1.1 e 4.1.2. Verifica-se que ambos os requisitos funcionais foram cumpridos, pois o sistema possui um terminal interativo, e também possibilita a consulta ao histórico de sessões. Quanto aos requisitos não-funcionais, os mesmos são listados e verificados abaixo:

- O sistema deve possuir controle de acesso, isto é, apenas determinados usuários poderão acessar o modo de terminal.
  - Conforme comentado durante o capítulo 4, o controle de acesso já é uma funcionalidade existente no AgrOn Acesso Remoto, e não precisou ser construído como parte do sistema, apenas integrado à aplicação. O acesso ao modo de terminal está disponível apenas aos usuários que possuem registro na tabela "*special_user*". Portanto, esse requisito foi cumprido.
- O sistema deve ser composto por uma aplicação web baseada nos serviços da AWS e implementada utilizando o *framework* Angular, e um agente embarcado responsável pela comunicação com o display.
  - O capítulo 5 apresenta os detalhes de implementação do sistema, e é possível verificar o cumprimento do requisito, pois as restrições de tecnologia foram respeitadas para garantir a integração com o sistema já existente.
- A autenticação no lado do display não deve estar associada à senha de SSH.
  - A autenticação do lado do display é realizada por meio do próprio serviço *iot_client*, que obtém credenciais para acesso aos serviços da AWS. Além disso, apenas os usuários autorizados, por associação prévia ou código de acesso, podem utilizar a ferramenta, o que significa que esse requisito também foi cumprido, pois não é mais necessário saber a senha de SSH

para acessar o display, porém ainda existe um mecanismo de autenticação que garante a segurança da conexão.

- O sistema deve utilizar os bancos de dados e estruturas já implementados para o AgrOn Acesso Remoto.
  - De forma similar ao requisito de restrição de tecnologias, este requisito pode ser considerado cumprido, pois a ferramenta desenvolvida foi integrada ao sistema já existente, incorporando suas funcionalidades.
- O sistema deve interagir com o display de forma que a performance das demais aplicações embarcadas não seja afetada negativamente pela sua existência.
  - Os resultados de performance vistos na Figura 38 indicam que o sistema se comportou de forma similar ao *Shellhub*, que não acrescentou *overhead* significativo à aplicação principal do sistema. De acordo com os desenvolvedores do time de embarcado, é necessário uma margem de ao menos *20%* de memória livre para comportar cenários de alto uso, e essa margem foi respeitada durante os testes. Portanto, esse requisito também pode ser considerado cumprido.

A latência encontrada foi considerada média para sistemas de tempo real, e o impacto na usabilidade do sistema é imperceptível. Um problema foi detectado, no entanto, em que algumas mensagens demoram mais tempo do que outras para serem processadas pela função Lambda, cuja alocação de recursos é dinâmica, o que faz com que caracteres digitados no terminal possam ser recebidos fora de ordem pelo display. Em discussão com o time de desenvolvimento, uma das soluções levantadas para esse problema seria a introdução de uma fila FIFO utilizando o serviço SQS *(Simple Queueing Service)*, mas isso aumentaria os custos da infraestrutura, e como as ocorrências desse *bug* são raras, optou-se por manter a implementação no estado atual.

A ferramenta está disponível apenas no ambiente de desenvolvimento no momento, e deve passar por testes, validações e melhorias adicionais antes de sua liberação no ambiente de produção.

7 Conclusão

Este documento apresentou o desenvolvimento de um sistema centralizado para acesso remoto no modo de terminal aos displays da Hexagon, que foi integrado ao sistema principal de acesso remoto da empresa, o Agron Acesso Remoto. Os testes de usabilidade e performance indicam que os requisitos estabelecidos foram cumpridos.

Com o lançamento do novo sistema, é esperado que a equipe de suporte apresente maior eficiência ao lidar com solicitações onde o acesso remoto aos displays é necessário, pois os problemas oriundos das limitações encontradas na ferramenta interna utilizada anteriormente foram solucionados. Imagina-se também que, em um futuro próximo, essa nova funcionalidade possa ser útil não só aos membros da equipe de suporte da Hexagon, mas também a outras equipes e parceiros externos que auxiliam os clientes no suporte em campo, sendo necessária a elaboração de uma documentação que possa ser empregada em treinamentos.

Como sugestão de desenvolvimentos futuros, é possível citar a investigação e resolução do problema encontrado onde alguns caracteres são recebidos fora de ordem pelo display, e a adição de uma ferramenta para transferência de arquivos, que poderia ser utilizada para o download de arquivos de diagnóstico online, por exemplo.

Em geral, avalia-se que o desenvolvimento deste projeto representa uma melhoria significativa nos processos de suporte da Hexagon, se convertendo em um diferencial na resolução de problemas em campo e, por consequência, gerando um aumento da confiança nas soluções da empresa por parte dos clientes.

REFERÊNCIAS

AMAZON WEB SERVICES. **Amazon API Gateway | Gerenciamento de APIs | Amazon Web Services**. [*S.l.*: *s.n.*], c2022a. Disponível em: https://aws.amazon.com/pt/api-gateway/.

AMAZON WEB SERVICES. **AWS IoT Core**. [*S.l.*: *s.n.*], c2022b. Disponível em: https://aws.amazon.com/pt/iot-core/.

AMAZON WEB SERVICES. **AWS Lambda**. [*S.l.*: *s.n.*], c2022c. Disponível em: https://aws.amazon.com/pt/lambda/.

AMAZON WEB SERVICES. **Banco de dados de chave-valor NoSQL rápido – Amazon DynamoDB**. [*S.l.*: *s.n.*], c2022d. Disponível em: https://aws.amazon.com/pt/dynamodb.

AMAZON WEB SERVICES. **Gerenciamento de acesso e identidade do cliente | Amazon Cognito**. [*S.l.*: *s.n.*], c2022e. Disponível em: https://aws.amazon.com/pt/cognito/.

AMAZON WEB SERVICES. **O que é AWS? Como funciona Amazon Web Services**. [*S.l.*: *s.n.*], c2022f. Disponível em: https://aws.amazon.com/pt/what-is-aws/.

AMAZON WEB SERVICES. **Recursos do Amazon RDS**. [*S.l.*: *s.n.*], c2022g. Disponível em: https://aws.amazon.com/pt/rds/features.

AMAZON WEB SERVICES. **Tutorial: Building a serverless chat app with a WebSocket API, Lambda and DynamoDB**. [*S.l.*: *s.n.*], c2022h. Disponível em: https://docs.aws.amazon.com/apigateway/latest/developerguide/websocket-api-chat-app.html.

AMAZON WEB SERVICES. **Tutorial: Criar uma API REST importando um exemplo**. [*S.l.*: *s.n.*], c2022i. Disponível em: https://docs.aws.amazon.com/pt_br/apigateway/latest/developerguide/api-gateway-create-api-from-example.html.

BARRETT, Daniel J; SILVERMAN, Richard E. **SSH, the Secure Shell: the definitive guide**. [*S.l.*]: "O'Reilly Media, Inc.", 2001.

DRUMOND, Claire. **Scrum — o que é, como funciona e por que é incrível**. [*S.l.*: *s.n.*], c2022. Disponível em: https://www.atlassian.com/br/agile/scrum.

HEXAGON. **About Us**. [*S.l.*: *s.n.*], c2022a. Disponível em: https://hexagon.com/about.

HEXAGON. **Agriculture**. [*S.l.*: *s.n.*], c2022b. Disponível em: https://hexagon.com.br/geospatial-solutions/hexagon-agriculture.

HEXAGON. **HxGN AgrOn ECU**. [*S.l.*: *s.n.*], c2022c. Disponível em: https://hexagon.com/products/hxgn-agron-ecu.

HEXAGON. **HxGN AgrOn Ti5**. [*S.l.*: *s.n.*], c2022d. Disponível em: https://hexagon.com/products/hxgn-agron-ti5.

HEXAGON. **HxGN AgrOn Track Controller**. [*S.l.*: *s.n.*], c2022e. Disponível em: https://hexagon.com/pt/products/hxgn-agron-track-controller.

HIMMELWRIGHT, Ryan. **Simple Reverse SSH Tunnels**. [*S.l.*: *s.n.*], ago. 2017. Disponível em: https://ryan.himmelwright.net/post/simple-reverse-ssh-tunnel.

KLUYVER, Thomas. **Ptyprocess Docs**. [*S.l.*: *s.n.*], 2014. Disponível em: https://ptyprocess.readthedocs.io/en/latest/.

MCCARTY, Scott. **Containers: Understanding the difference between portability, compatibility and supportability**. [*S.l.*: *s.n.*], jul. 2020. Disponível em: https://www.redhat.com/en/blog/containers-understanding-difference-between-portability-compatibility-and-supportability.

MELNIKOV, Alexey; FETTE, Ian. **The WebSocket Protocol**. [*S.l.*]: RFC Editor, dez. 2011. RFC 6455. (Request for Comments, 6455). DOI: 10.17487/RFC6455. Disponível em: https://www.rfc-editor.org/info/rfc6455.

OLDCOMPUTR.COM. **Digital VT100 (1978)**. [*S.l.*: *s.n.*], dez. 2015. Disponível em: https://www.oldcomputr.com/digital-vt100-1978/.

OPENSOURCE.COM. **6 exciting new ShellHub features to look for in 2021**. [*S.l.*: *s.n.*], mai. 2021. Disponível em: https://opensource.com/article/21/5/shellhub-new-features.

RED HAT. **O que é a Internet das Coisas (IoT)?** [*S.l.*: *s.n.*], jan. 2019. Disponível em: https://www.redhat.com/pt-br/topics/internet-of-things/what-is-iot.

RICHARDSON, Leonard; RUBY, Sam. **RESTful web services**. [*S.l.*]: O'Reilly Media, Inc., 2008.

RODRIGUES, Mônica. O Poder de uma SPA. **Revista Programar**, v. 53, n. 5, p. 54–57, 2016. Disponível em: https://www.revista-programar.info/artigos/o-poder-de-uma-spa/.

SADALAGE, Pramod J; FOWLER, Martin. **NoSQL distilled: a brief guide to the emerging world of polyglot persistence**. [*S.l.*]: Pearson Education, 2013.

TEIXEIRA, Silvana. **Cana-de-açúcar - Colheita**. [*S.l.*: *s.n.*], c2022. Disponível em: https://www.cpt.com.br/calendario-agricola/cana-de-acucar-colheita.

THE LINUX FOUNDATION. **The Linux Foundation Announces Yocto Project Steering Group and Release 1.0**. [*S.l.*: *s.n.*], c2011. Disponível em: https://www.linuxfoundation.org/press/press-release/the-linux-foundation-announces-yocto-project-steering-group-and-release-1-0.

THE LINUX FOUNDATION. **The Yocto Project Overview and Concepts Manual**. [*S.l.*: *s.n.*], c2018. Disponível em: https://docs.yoctoproject.org/2.5/overview-manual/overview-manual.html.

WAZLAWICK, Raul. **Análise e design orientados a objetos para sistemas de informação: Modelagem com UML, OCL e IFML**. [*S.l.*]: Elsevier Brasil, 2016.

WING, Dan. Network Address Translation: Extending the Internet Address Space. **IEEE Internet Computing**, v. 14, n. 4, p. 66–70, 2010. DOI: 10.1109/MIC.2010.96.

XORIANT. **WebSocket – Web is Stateful Now**. [*S.l.*: *s.n.*], c2022. Disponível em: https://www.xoriant.com/blog/websocket-web-is-stateful-now.