UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MARCUS VENICIUS PINA CAVALCANTI

XENON: UM SISTEMA PARA GERENCIAMENTO DE ACESSO DE AUTOMÓVEIS POR RECONHECIMENTO DE IMAGENS

GUARAPUAVA

2022

MARCUS VENICIUS PINA CAVALCANTI

XENON: UM SISTEMA PARA GERENCIAMENTO DE ACESSO DE AUTOMÓVEIS POR RECONHECIMENTO DE IMAGENS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Roni Fabio Banaszewsk

Coorientador: Prof. Me. Paulo André Filipak

GUARAPUAVA

2022

MARCUS VENICIUS PINA CAVALCANTI

XENON: UM SISTEMA PARA GERENCIAMENTO DE ACESSO DE AUTOMÓVEIS POR RECONHECIMENTO DE IMAGENS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 28/março/2022

Prof. Roni Fabio Banaszewski
Doutor
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

---

Prof. Hermano Pereira
Doutor
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

---

Prof. Guilherme da Costa Silva
Mestre
Universidade Tecnológica Federal do Paraná - Campus Guarapuava

GUARAPUAVA

2022

Dedico a todos os envolvidos, professores, alunos, dentre muitos outros da qual fizeram um esforço para este trabalho seja executado. Além de toda a comunidade acadêmica da UTFPR.

AGRADECIMENTOS

Agradeço a todos os envolvidos neste trabalho. Primeiramente, agradeço ao Professor Dr. Diego Marczal pelas dicas para o melhoramento do meu trabalho e também ao discente Pedro Ida pela ajuda. Em especial, agradeço ao orientador deste trabalho, o Professor Dr. Roni Fabio Banaszewski por toda a atenção, revisões e contribuições para com este trabalho.

É dito que um programa de computador aprende a partir de uma experiência E com respeito a uma classe de tarefas T e medida de desempenho P, se seu desempenho nas tarefas em T, segundo a medida P, melhora com a experiência E. (Mitchell, 1997).

RESUMO

O surgimento de algoritmos de aprendizagem de máquina proporcionou novos campos de estudos na computação e diversas aplicabilidades no mundo real. Muitas destas aplicabilidades estão relacionadas à visão computacional, que estuda o processamento de imagens para interpretação e avaliação das informações contidas nela. Como exemplo de tais aplicabilidades estão os Sistemas de Reconhecimento Automatizado de Placas (ALPR), os quais tem a finalidade de reconhecer os caracteres que integram uma placa veicular. A tecnologia ALPR é largamente utilizada em sistemas de tráfego e monitoramento de estacionamentos. Neste sentido, o corrente trabalho consiste no desenvolvimento de uma aplicação web integrada a uma aplicação ALPR implementada em Raspberry no contexto de Internet das Coisas para auxiliar no gerenciamento de acesso de veículos ao estacionamento da Universidade Tecnológica Federal do Paraná, campus Guarapuava, por meio da tecnologia ALPR. Esta aplicação é denominada Xenon.

Palavras-chave: alpr; ocr; alpr em raspberry.

ABSTRACT

The emergence of machine learning algorithms has provided new fields of studies in computing and diverse applicability in the real world. Many of these applications are related to computer vision, which studies image processing to interpret and evaluate the information contained therein. As an example of such applications are the Automated License Plate Recognition Systems (ALPR), which are intended to recognize the characters that make up a license plate. ALPR technology is widely used in traffic and parking monitoring systems. In this sense, the current work consists in the development of a web application integrated to an ALPR, implemented in Raspberry in the context of Internet of Things to assist in the management of vehicle access to the parking lot of the Federal University of Technology – Paraná - Guarapuava Campuses. This application is called Xenon.

Keywords: alpr; ocr; alpr on raspberry.

LISTA DE FIGURAS

LISTA DE FOTOGRAFIAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Abreviaturas

ai.	Inteligência Artificial
art.	Artigo
cap.	Capítulo
cm.	centímetro
gb.	giga byte
gnd.	Ground
m.	metros
mm.	milímetros
sec.	Seção

Siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
ALPR	Automatically Recognize License Plates
API	Application Programming Interface
CI-CD	continuos integration e continuos delivery
CM	centímetro
CPU	Central Processing Unit
CSI	Camera Serial Interface
CSS	Cascading Style Sheets
DERAC-GP	Departamento de Registros Acadêmicos
DIRGEP	Diretoria de Gestão de Pessoas
DOM	Document Object Model
GPIO	General-purpose input/output
HTML	Hyper Text Markup Language

IETF	Internet Enrineering Task Force
IoT	Internet of Things
JDBC	Java Database connectivity
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JSTL	Java Server Pages Standart Template Library
JVM	Java Virtual Machine
JWT	Json Web Token
MS	milissegundos
OCR	Optical Caracters Recognition
ORM	Object Relational Mapper
PIR	Passive Infrared Sensor
PWA	Progressive Web App
RAM	Random Access Memory
RDBMS	Relational Database Management System
REST	Representation State Transfer
RFID	Radio Frequency Identification
SEO	Search Engine Optimization
SoC	system-on-chip
SPA	Single Page Application
SQL	Structured Query Language
SSD	Solid State Drive
TCP	IP ransmission Control Protocol/Internet Protocol
URI	Uniform Resource Identifiers
URL	Uniform Resource Locator
UTFPR	Universidade Tecnológica Federal do Paraná

VCC	voltage direct current
WoT	Web of Things
XML	Extensible Markup Language
YAML	YAML é um acrónimo recursivo que significa "YAML Ain’t Markup Language"

SUMÁRIO

1 INTRODUÇÃO

No CAMPUS de Guarapuava da Universidade Tecnológica Federal do Paraná (UTFPR) há um fluxo diário elevado de pessoas e veículos, principalmente em horários que antecedem os períodos das aulas. O acesso de pessoas e veículos nas dependências do campus é controlado por funcionários de uma empresa de segurança terceirizada. Estes funcionários são responsáveis por permitir ou não o acesso de veículos ao estacionamento, além de recepcionar os visitantes.

A tomada de decisão para a autorização de acesso ao campus se dá através da verificação da existência de um adesivo com características particulares colado nos para-brisas dos veículos. Este adesivo contém o logo da UTFPR e é disponibilizado a alunos, técnicos administrativos e professores com o propósito de facilitar a identificação. Para cada uma das três categorias de motoristas é definida uma cor para os adesivos: o azul é usado para identificar terceirizados, o verde para alunos e o amarelo para professores e técnicos administrativos. Portanto, por meio das cores, o funcionário consegue extrair uma informação a mais para tomada de decisão. Porém, ele não tem dados suficientes para identificar o motorista.

Além disso, este processo de autorização de acesso traz consigo alguns problemas, tais como: o desgaste prematuro do adesivo se colado em local externo, problema muito comum em motocicletas; posicionamento do adesivo em um local de difícil visibilidade ao funcionário de segurança, que ocorre quando o adesivo é apenas deixado no painel do carro e mesmo o não uso do adesivo, quando o funcionário já conhece o veículo do servidor. Ademais, um outro problema ocorre quando o motorista vende o veículo e esquece de remover o adesivo. Com isso, o novo dono do carro poderia ter livre acesso ao campus sem necessariamente fazer parte da comunidade universitária. Neste sentido, este controle baseado em adesivos também se apresenta falho por causa da facilidade de falsificação de tais adesivos, levando ao acesso ao estacionamento de qualquer indivíduo que pactue com tal fraude.

Uma solução seria criar uma base de dados dos motoristas com permissão de acesso ao campus e também de seus veículos identificados por suas placas. A equipe de segurança tendo acesso a esta base de dados de uma forma simples e rápida, poderia ter maior poder para a tomada de decisão. Esta rapidez demanda a implementação de um sistema constituído por uma câmera de imagem e da tecnologia de Reconhecimento Automatizado de Placas (ALPR) de veículos. Com isso, as informações dos motoristas poderiam ser pesquisadas na base de dados e apresentadas automaticamente ao funcionário imediatamente a chegada de um veículo na portaria do campus. A identificação da placa por uma solução inteligente de tratamento de imagens evita esforços do funcionário para ler a placa do veículo e digitar no sistema, atividade que pode prejudicar o bom fluxo de veículos por demandar tempo e ser tendente a erro de digitação.

Com esta solução, não haveria mais a necessidade de depender de adesivos, uma vez que a placa serviria como credencial de acesso. Também, caso o motorista venda o veículo ou

termine o vínculo com a UTFPR por algum motivo, bastaria atualizar esta informação na base de dados. Neste sentido, quando um palestrante externo for convidado ao campus, bastaria cadastrá-lo nesta base de dados juntamente com os dados de seu veículo para este ter livre acesso ao estacionamento do campus.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um sistema com reconhecimento de placas de automóveis para controlar o acesso destes ao campus Guarapuava da UTFPR.

1.1.2 Objetivos Específicos

- Implantar um ALPR em uma plataforma *Raspberry*;
- Construir uma aplicação web para receber os dados do ALPR;
- Construir uma aplicação para integração com o OpenALPR;
- Treinar o algorítimo ALPR para leitura das placas de automóveis do Brasil;

1.2 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado da seguinte maneira: No capítulo 3 é apresentado um estudo sobre os sistemas correlatos, incluindo alguns sistemas que direta ou indiretamente servem ao mesmo propósito. Já no capítulo 2 é apresentada a fundamentação teórica, da qual, norteará o trabalho, não obstante, as tecnologias utilizadas. No capítulo 5 é apresentada a análise e projeto do sistema. Em consonância, no capítulo 6 é apresentado o processo de desenvolvimento do sistema de acordo com a Metodologia de desenvolvimento escolhido. Na sequência, no capítulo 7 são apresentados os resultados obtidos nos testes da solução proposta. Por fim, no capítulo 8 são apresentadas as considerações finais do trabalho, dando uma perspectiva sobre os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Para construir um sistema que seja robusto e escalável, há a necessidade de planejamento e organização, mas não só na abordagem arquitetural, mas também em termos de tecnologias e ferramentas utilizadas e principalmente de um gerenciamento eficaz do projeto. Estudar os conceitos teóricos para resolver um problema é um processo intuitivamente natural, pois traça as fronteiras entre os estudos já estabelecidos e o que deverá ser inovado para uma solução ao nível do problema. Neste sentido, o corrente trabalho baseia-se em estudos acadêmicos e de tecnologias do mercado para propor uma solução.

O Sistema Xenon é composto por duas aplicações: uma aplicação de reconhecimento de placas com ALPR e uma aplicação web para consumir os dados da aplicação de reconhecimento. Mais precisamente, a aplicação de reconhecimento é responsável por fazer o processo de reconhecimento utilizando-se de aprendizagem de máquina, ao passo que a aplicação web é subdividida em duas partes: frontend e backend, que em conjunto será responsável por exibir, salvar e processar os reconhecimentos provenientes da aplicação de reconhecimento. Ambas as aplicação serão executadas em ambientes diferentes.

Basicamente, o capítulo está estruturado da seguinte maneira: na seção 2.1 será apresentado os conceitos utilizados para compor ambas as aplicações e o formato de transporte de dados. Na seção 2.2 serão apresentados os conceitos teóricos que norteiam a aplicação de reconhecimento, tal como o treinamento do algoritmo para reconhecimento de placas pelo uso das tecnologias de aprendizagem de máquina e conceitos da tecnologia de Internet das Coisas, que por sua natureza distribuída e inteligência agregada, está fortemente relacionada ao corrente trabalho.

2.1 ARQUITETURA REST

Fielding e Taylor (2000) em sua dissertação de doutorado, propôs um estilo de arquitetura denominada REST. Este estilo propõe que os lados cliente e servidor fiquem separados e independentes, alcançando a separação de responsabilidades. O estilo foi baseado em um estilo arquitetural denominado Cache-Servidor sem Estado, no qual um serviço não depende de estados de sessão para controlar a resposta da requisição. Assim, uma requisição necessariamente deve conter as propriedades necessárias para completar o seu processo. Assim, o servidor não se responsabiliza pelas informações necessárias, deixando a cargo do cliente conhecer todas as informações. Outra característica evidente ao estilo REST é a interface uniforme, padronizando as chamadas através do uso do protocolo HTTP e seus métodos, e códigos de resposta.

O padrão REST determina como deve ser realizada a transferência de dados, que representa e corresponde ao conjunto de valores que representa uma determinada entidade em um dado momento. Essa transmissão de estados se dá a partir da especificação de parâmetros,

em alguns casos chamados de restrições. Quando isso ocorre, o que se obtém são serviços escaláveis, de fácil modificação e manutenção, e que apresentam boa performance, tornando esses serviços adequados a serem utilizados através da Internet.

RESTful é o termo para determinar a capacidade de uma aplicação web aplicar os princípios que o REST propõe. Nos serviços RESTful, tanto os dados quanto as funcionalidades são considerados recursos e ficam acessíveis aos clientes através da utilização de URIs (*Uniform Resource Identifiers*). Um recurso é acessado por um endereço na web que integra a identificação tanto do servidor no qual a aplicação está hospedada quanto da própria aplicação.

Os recursos disponíveis em uma aplicação RESTful podem ser acessados ou manipulados a partir de um conjunto de ações ou métodos predefinidos no protocolo HTTP. Conforme Vernon (2016), as operações possibilitam alterar (PUT ou PATCH), ler (GET), criar (POST) e apagar (DELETE) recursos.

O *Javascript Object Notation* (JSON) refere-se a um formato de transporte de dados usado na web, principalmente em conjunto com aplicações que seguem as diretivas da arquitetura REST. Com base na especificação *JavaScript Programming Language Standard*, ECMA-262, o JSON nada mais é do que um objeto representado em forma de texto, isso traz consigo a independência de linguagem de programação, o que proporciona uma forma ideal de intercâmbio de dados entre sistemas independentes. Com uma estrutura de chave e valor ou uma lista ordenada de dados em sequência, da qual a maioria das linguagens modernas consegue interpretar, este formato torna-se ideal para transferência de dados pela web, principalmente por ser significativamente menos verboso do que o tradicional XML.

2.2 TECNOLOGIAS DA APLICAÇÃO DE RECONHECIMENTO DE PLACAS

Nesta seção será explanado sucintamente as tecnologias e conceitos teóricos utilizados para compor a aplicação de reconhecimento de placas. Mais precisamente, na subseção 2.2.1 são apresentados os conceitos de inteligencia artificial, aprendizagem de máquina, visão computacional e uma explicação sobre ALPR. Por sua vez, na subseção 2.2.2 são apresentados os conceitos de Internet das Coisa(IoT) e Web das Coisas (WoT). Por fim, na subseção 4 são apresentadas as tecnologias utilizadas para compor a aplicação de reconhecimento de placas.

2.2.1 Reconhecimento de Placas de Veículos

A Inteligência Artificial(IA) tem como propósito o aprimoramento cognitivo dos agentes, além de processamento de linguagem natural, representação do conhecimento, raciocínio automatizado, visão computacional e robótica. Como explica RUSSEL Stuart; NORVIG (2013):

> Processamento de linguagem natural, para permitir que eles se comuniquem com sucesso em um idioma natural, representação de conhecimento para armazenar o que sabe ou ouve; raciocínio automatizado para usar as

> informações armazenadas com finalidade de responder a perguntas e tirar novas conclusões; aprendizado de máquina para se adaptar a novas circunstâncias e para detectar e extrapolar padrões; [...] visão computacional para receber objetos e robótica para manipular objetos e movimentar-se.

Segundo RUSSEL Stuart; NORVIG (2013), agente é o termo dado para um componente computacional que percebe seu ambiente através de sensores, e interage com este ambiente via atuadores. Um agente através da aprendizagem pode melhorar o seu comportamento baseando-se em alguns fatores, tal como no conhecimento prévio do agente e *Feedback* de aprendizagem. Segundo RUSSEL Stuart; NORVIG (2013), a Inteligência Artificial se baseia em algumas linhas computacionais de raciocínio: indutiva, analítica e dedutiva. Neste sentido, pode-se constatar que um feedback na aprendizagem é indutivo, ou seja, o agente coleta informações, ao passo que explora seu ambiente.

Mais precisamente, há três categorias para as tarefas de aprendizagem de máquina: aprendizagem supervisionada, não supervisionada e por reforço. Estas categorias serão apresentadas a seguir.

Aprendizagem não supervisionada: o agente aprende através da detecção de grupos de exemplos, ou seja, por meio de entradas de dados úteis. Logo, o agente analisa um conjunto de dados e realiza através de probabilidade o agrupamento de dados similares. Assim, o agente aprende baseando-se no agrupamento de resultados positivos e negativos sem nunca ter rotulado exemplos dos mesmos (RUSSEL STUART; NORVIG, 2013).

Aprendizagem supervisionada: o agente recebe um conjunto de dados conhecido, ou seja, um conjunto de entrada e saída. As entradas são as percepções do agente e as saídas são representadas por um valor explícito pelo instrutor do agente. Takatuzi (2017) explica que neste tipo de aprendizado “é fornecido ao algoritmo de aprendizagem um conjunto de dados de treinamento nos quais a classe associada a cada dado é conhecida por um supervisor externo.”

Aprendizagem por reforço: o agente aprende à medida que recebe recompensa ou punição. Neste tipo de aprendizagem, o agente decide pelas próximas ações baseando-se no histórico de ações realizadas e no reforço que recebeu (RUSSEL STUART; NORVIG, 2013).

A aprendizagem de máquina proporcionou novos campos de estudos, tal como a Visão Computacional. Esta estuda o processamento de imagens a fim de interpretar e avaliar as informações contidas nela.

Basicamente, o processamento de imagens se assemelha bastante com a visão computacional. Contudo, algumas diferenças sutis podem ser aferidas por meio do estudo de conceitos estabelecidos na literatura, como explica Marengoni e Stringhini (2009): "O processamento ou tratamento de imagens é um processo onde a entrada do sistema é uma imagem e a saída é outra imagem com uma melhor qualidade, ao passo que visão computacional utiliza-se de conceitos de aprendizagem de máquina para processar uma imagem melhorada por algoritmos de processamento de imagens a fim de emular a visão humana".

Este paralelo entre visão computacional e processamento de imagem é importante pelo fato do processo de visão computacional geralmente ser iniciado pelo processamento de uma

imagem a fim de melhorar a qualidade da mesma (por meio da redução de ruído, modificação de cores, ajustes no contraste e iluminação, entre outras melhorias) na intenção de facilitar a identificação de algum objeto na imagem pelo agente da visão computacional.

Conforme (SCURI, 1999), um processamento de imagem pode ser classificado em duas classes: por escopo ou por resultado. Cada uma é formada por técnicas e processos distintos, entre as quais, seguem um fator em comum: a qualidade. Também, (SCURI, 1999) explica que existem duas divisões no contexto de qualidade em processamento de imagem: fidelidade e inteligibilidade. Na fidelidade, o processo é voltado para aproximar a imagem processada da imagem original ou de um padrão estipulado que a melhor represente. No segundo caso, preocupa-se com a informação extraída da imagem, seja pelo olho humano ou por algum processamento.

Em suma, o uso do processamento de imagem tem foco na qualidade da imagem e a visão computacional utiliza-se da inteligibilidade para reconhecimento de objetos ou rastreamento. Ambos os processos possuem uma interação íntima na área de Inteligência Artificial. O processo de reconhecer um objeto é uma das principais áreas de visão computacional, a qual está relacionada ao estudo de padrões.

Segundo (MARENGONI; STRINGHINI, 2009), o processo de rastreamento é o reconhecimento de um padrão em uma finita sequência de imagens. Mais precisamente, o processo de rastreamento está atrelado ao conhecimento sobre o movimento do objeto que está sendo rastreado no intuito de minimizar a busca dentre as imagens em uma sequência. Um outro processo importante é o de segmentação. O processo de segmentação tem como função a divisão da imagem por região ou objetos distintos. O algoritmo busca características distintas do objeto, cor, tonalidade, *pixels* das quais têm similaridades e podem ser conectados.

Com base nos parágrafos anteriores podemos perceber similaridades entre processamento de imagem e visão computacional. Os processos de reconhecimento de objetos e rastreamentos de objetos em sequência de imagens são processamentos de imagens onde um agente utiliza de técnicas de aprendizagem de máquina supervisionada para reconhecer padrões nas imagens, assim como o rastreamento. Enfim, o processamento de imagem tem como finalidade a qualidade holística da imagem e a visão computacional emprega o processamento de imagem como um ente processo para sua finalidade. Na visão computacional, por meio de técnicas de aprendizagem de máquina supervisionada, foi possível a descoberta da tecnologia de Reconhecimento Ótico de Caracteres (OCR da sigla em inglês *Optical Character Recognition*). Por sua vez, num avanço de aplicação do OCR, surgiu o Sistema de Reconhecimento Automatizado de Placas (ALPR), o qual tem a finalidade de reconhecer os caracteres que integram uma placa veicular.

Um sistema ALPR (*Automatic Licence Plate Recognition*) consiste em um processo de transformação de uma entrada complexa composta por imagem ou *stream* de vídeo em uma saída simples de texto, ou mais precisamente, nos caracteres de uma placa veicular. No ALPR, são necessárias ao menos 3 etapas para um reconhecimento de placas de veículos: locali-

zação, segmentação dos caracteres e reconhecimentos dos caracteres. A etapa de localização consiste em fazer um rastreamento na imagem a fim de detectar uma possível região na imagem onde os caracteres que formam a placa poderão ser encontrados, importante mencionar que o agente conhece previamente o objeto buscado, através do aprendizado supervisionado. Após a localização dos caracteres que formam a placa na imagem, ocorre a segmentação de cada caractere da placa em objetos distintos. Por fim, na etapa de reconhecimento dos caracteres, os objetos segmentados são associados a caracteres.

2.2.2 Internet das Coisas e Web das Coisas

A Internet das Coisas ou *Internet of Things* (IoT), em inglês, se refere a uma classe de tecnologias que tem como objetivo conectar dispositivos à rede mundial de computadores. Cada vez mais surgem eletrodomésticos conectados à Internet e a outros dispositivos. Isto possibilita, por exemplo, fechaduras inteligentes que tem como função notificar o usuário quando forem abertas, ou até mesmo abrir remotamente. Devido a sua importância ao contexto tecnológico atual.

O termo IoT foi criado por Kevin Ashton em 1999 em uma apresentação na empresa Procter & Gamble, quando o autor discursava sobre uma nova ideia para RFID (*Radio Frequency Identification*, em inglês, ou Identificação por radiofrequência) (ASHTON, 2009). Entretanto, o conceito já era discutido desde 1991 quando a Internet estava se popularizando. Neste contexto, o termo *The network is the computer* (a rede é o computador) foi primeiramente defendido pela empresa Sun com o significado de coleta de dados por dispositivos conectados à Internet. Por sua vez, Bill Joy, co-fundador da Sun, descreveu na sua apresentação no Fórum Econômico Mundial de Davos o conceito das *Six Web*. De forma resumida, o conceito é definido como:

1. A *Near Web*, aquela que está acessível pelo seu *desktop ou laptop*.
2. A *Here Web*, aquela que está disponível para seus dispositivos móveis.
3. A *Far Web*, aquela que está disponível em seus televisores.
4. A *Weird Web*, aquela em que você interage por voz.
5. A *B2B Web*, aquela em que máquinas interagem entre si.
6. A *D2D Web*, aquela que é a web dos sensores.

Mais precisamente, o item 6 foi apresentado em novembro de 1997 no Gartnet Group Symposiun, demonstrando a sua aplicabilidade. Neste evento, McNealy apresentou o Java Rings, precursor dos dispositivos IoT. McNealy explicou que o dispositivo poderia, entre muitas tarefas, abrir a porta do carro do usuário. Basicamente, o dispositivo era um cartão inteligente que era programado para identificar e interagir com o ambiente, comunicando-se via TCP/IP.

O principal mérito do Java Rings foi estimular o estudo para dispositivos móveis, por exemplo celulares, pois o dispositivo reforçava o conceito de mobilidade e a comunicação de objetos físicos que coletam dados ou trocam dados com outros dispositivos, pela Internet. Isto proporcionou nos dias de hoje diversas aplicabilidades, por exemplo: casas inteligentes, monitoramento remoto e em tempo real de pacientes e carros autônomos.

Inúmeras possibilidades podem surgir quando dispositivos são conectados à Internet. Estes são conceitualmente denominados objetos inteligentes ou Smart Objects em inglês. Em suma, um objeto inteligente deve necessariamente conter as seguintes características:

1. Unidade de processamento;
2. Unidade de memória;
3. Capacidade para comunicação entre outros dispositivos;
4. Sensores ou atuadores;

Nos anos de 2000, iniciou-se uma discussão a respeito destes dispositivos, que por natureza podem ter os mais variados propósitos e interfaces de comunicação, como explicado em França *et al.* (2011): “a maioria dos objetos são conectados à Internet (e algumas vezes a Web) utilizando softwares e interfaces proprietárias, o que torna onerosa a criação de aplicações”. A complexidade pode aumentar com a utilização de protocolos e interfaces específicas. Conforme o autor França *et al.* (2011): "é necessário utilizar uma linguagem comum aos diferentes dispositivos”.

Para solucionar o problema, a WoT propõe utilizar-se de protocolos Web na tentativa de criar uma linguagem comum para comunicação entre dispositivos físicos no meio digital. Com isso, dá a possibilidade para estes dispositivos executarem um serviço para ser utilizado em diferentes domínios de aplicações, em outras palavras, incentiva o reuso. Para isso, a WoT faz uso do protocolo HTTP para trafegar dados entre os dispositivos envolvidos conforme os princípios da arquitetura REST (*Representation State Transfer*), os quais são apresentados na seção 2.1.

> Segundo (FRANÇA *et al.*, 2011),a WoT emprega os princípios REST possibilitando duas abordagens. Na primeira abordagem, são implantados servidores Web embarcados em dispositivos inteligentes e as funcionalidades desses dispositivos são disponibilizadas na forma de recursos RESTful. Na segunda abordagem, quando um objeto inteligente não possui recursos de hardware suficientes para executar um servidor embarcado, é possível utilizar outro dispositivo como ponte para disponibilizar as funcionalidades do dispositivo inteligente na Web através de uma interface RESTful.

Estas abordagens permitem que os dispositivos exponham através dos recursos Web interfaces reutilizáveis e bem definidas. Com isso, o serviço pode ser consumido por diferentes aplicações.

3 SISTEMAS CORRELATOS

Neste capítulo são apresentadas as aplicações similares à aplicação desenvolvida neste trabalho. Basicamente, é apresentado um estudo sobre aplicações de mercado que diretamente ou indiretamente servem ao mesmo propósito. Mais precisamente, na seção 3.1 será dado um resumo sobre o sistema da empresa DBA, na seção 3.2 será explanado sucintamente o sistema de reconhecimento de placas da empresa Meerkat, na seção 3.3 será discutido de forma resumida o sistema AutoVu ALPR da empresa Genetec e, por fim na seção 3.4 conterá um estudo comparativo sobre os sistemas aqui mencionados.

3.1 SISTEMA DE IDENTIFICAÇÃO DA EMPRESA DE TECNOLOGIA DBA

O sistema da empresa DBA trabalha com *Optical Character Recognition* (OCR) em conjunto com ALPR para fazer o controle de acesso e rastreabilidade dos veículos nas dependências do estacionamento (DBA, 2021). Este sistema tem integração com alguns outros softwares de gerenciamento de estacionamento desprovidos das tecnologias de OCR ou ALPR. Entretanto, o ALPR da empresa DBA é um software proprietário, não dando a possibilidade de um cadastro aberto, deixando apenas a equipe de segurança adicionar os dados dos usuário e deixando inviável a prática de dar a responsabilidade para o usuário. Outrossim, por ser um software proprietário, sua customização torna-se inviável para as necessidade do campus.

3.2 DETECÇÃO, RECONHECIMENTO DE PLACAS DA EMPRESA MEERKAT

A Meerkat é uma empresa que trabalha com tecnologias da Visão Computacional. Mais precisamente, a empresa trabalha com as seguintes tecnologias: reconhecimento facial, reconhecimento de logomarcas e reconhecimento de placas (MEERKAT, 2021). O ALPR da empresa funciona com respostas em formato JSON advindas do reconhecimentos OCR. Contudo, o sistema é proprietário. A aquisição é através de licença de uso com tempo determinado.

3.3 AUTOVU ALPR

O AutoVu ALPR foi desenvolvido pela empresa Genetec. Ele consiste em um módulo que compõe o sistema Security Center da empresa, uma solução completa com videomonitoramento, rastreamento em tempo real, com aplicabilidade em diversos situações como controle de tráfego de cidades, controle de proteção a furtos de veículos (GENETEC, 2021). Por fim, sendo o AutoVu parte de um sistema robusto e complexo, pois a aquisição da solução implicitamente os dispositivos fixos ou imóveis para o monitoramento. Sendo assim inviável para o problema do acesso ao estacionamento do campus.

3.4 ESTUDO COMPARATIVO

Os sistemas apresentados são todos proprietários. O sistema da Genect é um sistema completo, sendo que o ALPR apenas compõe um módulo, ao passo que os sistemas da DBA e Meerkat são soluções da qual necessita de uma estrutura pronta, tal como câmera e servidor para instalar o sistema. Por fim, todos não são customizáveis e relativamente caros. Ao passo que a solução apresentado no referido trabalho é de baixo custo, utilizando equipamentos baratos, além de ser customizável e extensível, pois outros alunos que antecederam o autor do trabalho pode aprimorar ou incrementar novas funcionalidades.

4 TECNOLOGIAS UTILIZADAS E MÉTODO ÁGIL

Como mencionado anteriormente o sistema Xenon e composto por duas aplicações, web e reconhecimento, neste capitulo será apresentado as tecnologias envolvidas além de, o método e os processos para construção do sistema. Basicamente, na seção 4.1 será apresentando os componentes eletrônicos e as tecnologias utilizadas para aplicação de reconhecimento. É importante mencionar que esta é uma aplicação a parte, com ambiente de execução diferente da aplicação web para tanto, nesta seção são apresentados os componentes eletrônicos, o ambiente de execução NodeJS e por fim, a biblioteca OpenALPR que será utilizada para executar o reconhecimento propriamente dito. Seguindo, na seção 4.3 serão apresentados os conceitos de arquitetura e alguns padrões utilizados na área de desenvolvimento de aplicações para web. Na sequência, considerando que a aplicação proposta está arquiteturalmente organizada em três camadas: camada de apresentação, camada de negócios e camada de dados, são apresentadas algumas tecnologias usadas na camada de apresentação para construção de aplicações do lado cliente e complementarmente, são apresentadas as tecnologias usadas na camada de negócios para construção de aplicações do lado servidor e ainda, por fim são apresentadas as tecnologias para armazenamento de dados.

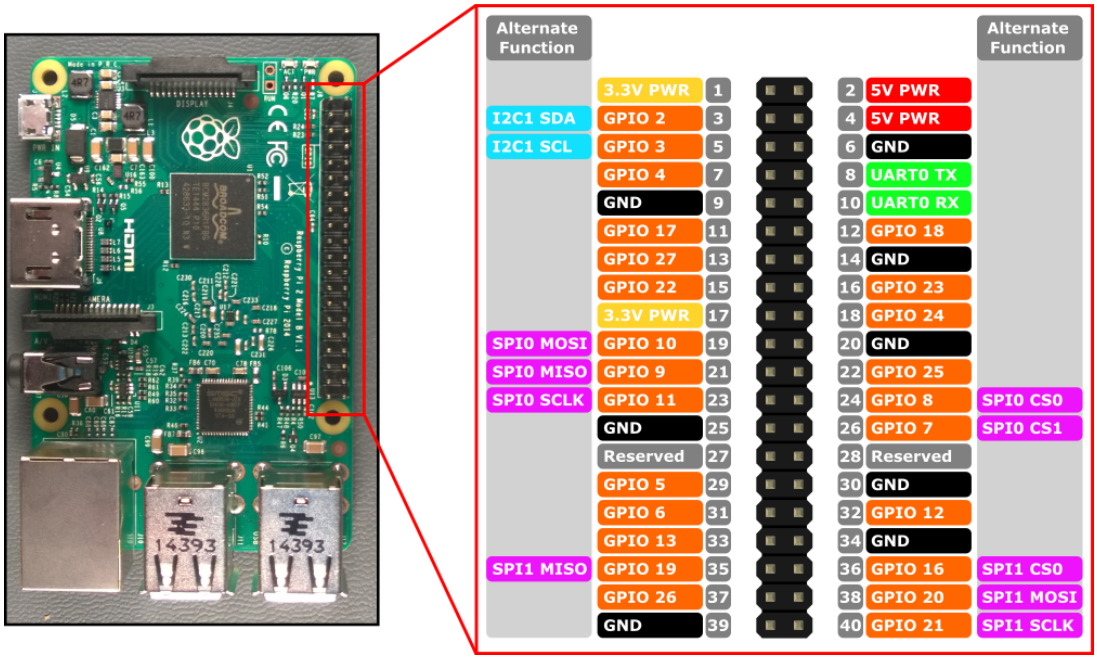
Não menos importante, na seção 4.4 são apresentados alguns conceitos de gerenciamento das atividades do projeto usando práticas ágeis e finalmente, na seção 4.5 são apresentadas algumas ferramentas que auxiliaram no desenvolvimento do sistema.

4.1 TECNOLOGIAS DE RECONHECIMENTO

Em uma definição simplista, o Raspberry é uma plataforma de hardware embarcada muito popular em projetos de IoT e WoT. Criado em meados de 2014 no Reino Unido pela fundação Raspberry Pi, o seu objetivo principal era promover o ensino em Ciência da Computação básica em escolas, inclusão e empoderamento social. O Raspberry pode ser considerado um computador de pequeno porte provido com processador multimídia Broadcom BCM2835 Quad Core 64-bit ARMv8 Cortex-A53 1.2GHz tipo SoC (*system-on-chip* – em inglês, sistema em um chip) e 1GB de memória RAM. Isso significa que a grande maioria dos componentes do sistema, incluindo sua unidade central e as de processamento gráfico, assim como o hardware de áudio e de comunicações, está montada em um único componente. O Raspberry Pi não tem nenhum disco rígido na sua composição de hardware, em vez disso, utiliza um cartão de memória. A plataforma Raspberry ainda oferece portas GPIO (*general-purpose input/output*), ou seja, uma entrada de comunicação genérica digital da qual pode ser usada para comunicação entre outros dispositivos. Esta porta tem dois estados, zero e um, onde um é para ligada e zero desligada. Quando ligada, a tensão no pino da GPIO é de 3.3V, ao passo que quando desligada é 0v. Também, o Raspberry tem oito conexões de GND (*ground* em inglês) comumente chamado de neutro, além de conexões VCC (voltage direct current em

inglês) utilizado para alimentar circuitos de corrente continua, sendo duas conexão de 5 volts e duas conexões de 3.3 volts.(GPIO, 2021). A Figura 1 exemplifica as conexões.

Figura 1 – GPIO Raspberry

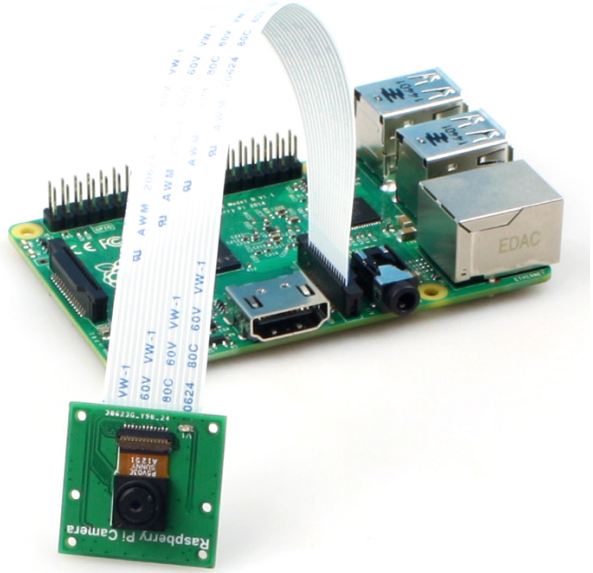


Fonte: microsoft.docs (2021).

O Módulo de câmera OV5647 da OmniVision, apresentado na Figura 2, foi utilizado neste referido trabalho. Este módulo pode ser aplicado para reconhecimento de imagens e rastreamento de imagens, conectando-se com a conexão CSI(*Camera Serial Interface*) do Raspberry, como demostrado na documentação CAM (2021). Ainda, ele pode ser utilizado para gravar vídeo de alta definição com 1080P e tirar fotos de 2592x1944. A sua lente é f=3.6, f/2.9, onde o **f** define a abertura do diafragma da câmera, ou seja a quantidade de luz que incide sobre a lente. Este valor é inversamente proporcional à abertura, sendo assim quanto maior o número, maior é a abertura. O ângulo de visão é de 54 x 41 graus e o campo de visão é de 2.0 x 1.33m em 2m. A seguir, são apresentados alguns detalhes técnicos do módulo, encontrado na documentação OKDO (2021):

1. Tamanho do Pixel: 1.4x1.4 hmm;
2. Abertura: 2.9;
3. Distância focal: 3.29;
4. FOV: 65 graus;
5. Foco fixo: 1 m até ao infinito;
6. Full-frame SLR lente equivalente: 35mm;

Figura 2 – Modulo de Câmera OV5647



Fonte: Jackson (2015).

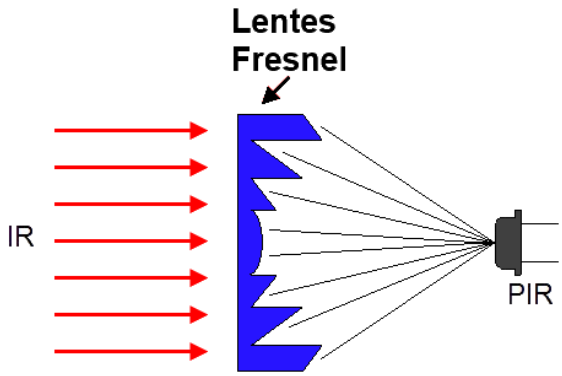
O Sensor de Movimento PIR (*Passive Infrared Sensor*) DYP-ME003 foi utilizado para compor o hardware da aplicação de reconhecimento. Este sensor consegue detectar o movimento de objetos que estejam em uma área de até 7 metros. Caso algo se movimente nesta área, o sensor é ativado. Ele é composto internamente por duas faixas com material sensível ao infravermelho. Na parte externa, possui uma lente fresnel filipeflop (2021). Ele conecta a uma fonte de 5v ao GND e VCC, além da conexão de dados que recebe o sinal de saída que será um (1) indicando movimento ou zero (0), indicando nenhuma movimentação. O sensor é apresentado na Figura 3.

Figura 3 – Sensor PIR

Fonte: filipeflop (2021).

Quando há variação na detecção do sinal infravermelho entre essas duas faixas de material sensível, a saída é acionada por um determinado tempo. A lente fresnel tem a função de ampliar o campo de visão do sensor, condensando a luz em um único ponto filipeflop (2021), como apresentado na Figura 4:

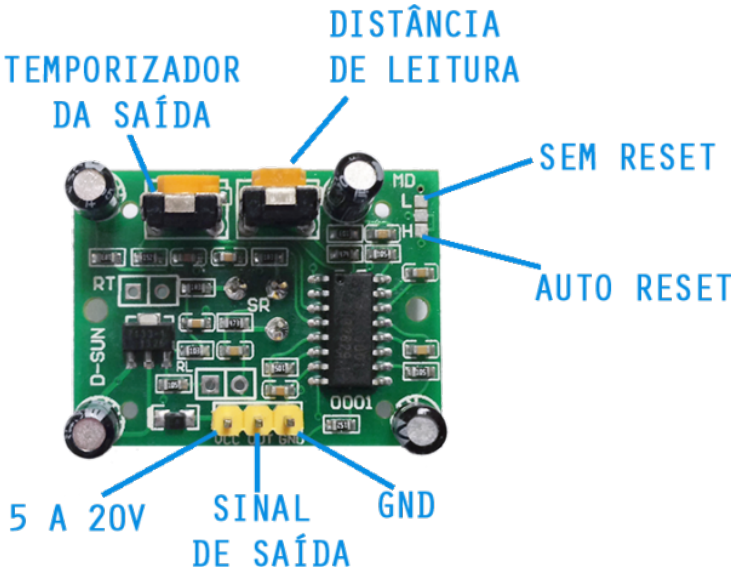
Figura 4 – Lente PIR



Fonte: filipeflop (2021).

Na Figura 5, são demonstradas as partes do sensor. Esta placa é composta por 2 pinos de alimentação e sinal, os potenciômetros para ajuste da sensibilidade que vai de 3 à 7 metros e tempo de acionamento da saída ajustável para até 5 à 200 segundos. Também, ela possui o *jumper* que controla o modo de operação do *trigger* (gatilho), sendo possível ajustar para um único sinal ou repetir o sinal.

Figura 5 – Sensor com partes eletrônicas



Fonte: filipeflop (2021).

Os componentes mencionados anteriormente, são de baixo custo, assim alcançando o objeto de propor uma solução barata. O Raspberry tem recursos computacionais suficientes para executar os softwares necessários para fazer o reconhecimento. Fazendo assim o isolamento dos sistema.

4.2 NodeJS

O NodeJS é um plataforma para *runtime* (ambiente de execução) single-thread para linguagem JavaScript. Isto quer dizer que o processo é executado em uma única *thread* no processador, trazendo benefícios de compartilhamento de recursos com toda a aplicação, como leitura e escrita em disco. Por essa natureza de ser um *runtime single-thread*, o processamento de requisições no NodeJS ocorre de forma assíncrona. Com isso, consegue-se mais leveza no processo, fazendo que ele utilize pouco recurso de hardware para ser executado. Ademais, o NodeJS utiliza-se a abordagem Orientado a Eventos e tem como base o motor de interpretação de código JavaScript, o *engine* V8 da Google. DOCS (2021) Neste sentido, como agente de reconhecimento detém recursos computacionais limitados, é o agente de reconhecimento que também terá uma aplicação web para enviar os reconhecimentos e receber a resposta via API RESTful. A plataforma NodeJS foi escolhida para tarefa, principalmente por utilizar pouco recurso computacional.

Uma outra tecnologia utilizada é a OpenALPR. Ela é uma biblioteca de reconhecimento de placa de licença automática de código aberto escrita em C++, que analisa imagens e fluxos de vídeo para identificar placas de carros. A saída é a representação de texto de quaisquer caracteres da placa de carros. Ela utiliza-se das técnicas de visão computacional para o processamento de imagem e aprendizagem de máquina para o trabalho. A biblioteca tem integração com diversas linguagens. O seu funcionamento consiste de forma resumida, como informa a documentação (TECHNOLOGY, 2017):

1. Busca na imagem uma possível região da placa;
2. Faz um tratamento na imagem para melhorar a detectação dos caracteres;
3. Tenta delimitar a região na imagem onde a placa está, buscando pela borda da placa;
4. Segmenta e isola os caracteres na região da placa;
5. Opcionalmente após encontrar os caracteres se o padrão da placa for informado, valida.

Na última fase de validação, a biblioteca se comporta da seguinte maneira: Caso informado um padrão de placa, por exemplo, o padrão antigo brasileiro **AAA-9999** (3 letras e 4 números), a biblioteca compara o padrão informado com as possíveis sequências de caracteres encontrados. Caso a comparação não seja bem sucedida, ela informa que não encontrou a placa correspondente ao padrão. Por fim, a biblioteca também provê o suporte a *Stream* de vídeo, o que facilita o desenvolvimento do sistema.

4.3 TECNOLOGIAS DA APLICAÇÃO WEB

Uma aplicação Web ou desktop comumente contém algumas responsabilidades, tais como: disponibilizar interação com usuário, processar essa interação e guardar o resultado das duas anteriores. Em resumo, um usuário incita um evento na aplicação, da qual processa esse evento, que guarda os dados resultantes deste evento. Distinguir as responsabilidades de um software é algo já intuitivo, tornando a distribuição das responsabilidades em camadas algo muito comum. Sobre o advento das aplicações da web em comparação às tradicionais aplicações desktop, Martin Fowler argumenta o seguinte:

> Com o advento da Web, de repente as pessoas passaram a querer instalar aplicações como cliente-servidor usando um navegador Web. No entanto, se toda a sua lógica de negócio estiver atrelada a um cliente rico, então toda ela precisará ser refeita para ter uma interface Web. Um sistema bem projetado, em três camadas, poderia simplesmente acrescentar uma nova camada de apresentação e estaria pronto. (FOWLER, 2009)

Neste sentido, surgiu o tradicional modelo em três camadas. Conforme Fowler (2009), ao separar uma aplicação em camadas consegue-se alguns benefícios, o mais evidente é o isolamento das partes da aplicação. Com isso, há a possibilidade de criar um ambiente para cada camada da aplicação IBM (2020), além de proporcionar melhor organização no desenvolvimento, pois as camadas podem ser desenvolvidas simultaneamente e até mesmo por equipes diferentes. Ainda, há a vantagem de proporcionar um ambiente escalável e seguro, pois cada camada pode ser ajustada conforme a demanda.

- Camada de **apresentação**: responsável por tratar da interação com usuário. Nesta camada serão apresentando os dados, ou seja, as possíveis ações que o usuário pode executar no software;
- Camada de **lógica de domínio**: nesta camada será realizado todo o processamento que envolve a lógica de negócio, ou seja, cálculos, validações, processamento das entradas e eventos do usuário;
- Camada de **dados**: esta camada será responsável pelo armazenamento dos dados, gerenciamento de transações e comunicação entre outros sistemas.

Um ponto interessante diz respeito aos termos referentes a arquitetura em camadas em inglês: *tier* e *layer*. Embora em tradução livre, estes termos sejam traduzidos para o português do Brasil como camadas, estes não são sinônimos. Conforme Fowler (2009), *tier* remete a uma separação física e *layer* enfatiza uma separação lógica. Isto leva ao pensamento que uma *tier* pode ser dividida em *layers* sem perder seu propósito. Portanto, o termo camada a partir deste ponto será substituído por *tier*, por se tratar de separação física, e *layer* para separação lógica, por exemplo, pacotes de componentes dentro de uma *tier*.

O conceito de isolamento da aplicação traz consigo algumas restrições evidentes para o bom funcionamento da arquitetura em três *tiers*, ou seja, a necessidade de comunicação entre as camadas para tráfego de dados. Em uma visão geral, a camada de lógica de domínio será a orquestradora das *tiers*, sendo ela a responsável por receber a entrada de dados ou eventos do usuário, realizar o processamento e, se necessário, invocar a *tier* de dados. Na sequência, este deve retornar o resultado para *tier* de apresentação.

Diante disto, nesta seção são apresentadas as tecnologias para a construção da aplicação web, as quais estão organizadas por *tier*. Basicamente, na subseção 4.3.1 são apresentadas as tecnologias referente a *tier* de lógica de negócio, na subseção 4.3.2 são apresentadas as tecnologias para a *tier* de apresentação, por fim, na seção 4.3.3, as tecnologias da *tier* de dados.

4.3.1 Camada de Lógica de Negócio

Nesta subseção são apresentadas as tecnologias utilizadas no desenvolvimento da aplicação Xenon no que concerne a *tier* de lógica de negócio. Mais precisamente, são apresentadas a linguagem Java e o *framework* do lado servidor chamado Spring Framework.

4.3.1.1 Java

A linguagem Java foi desenvolvida no início da década de 90 nos laboratórios da Sun Microsystems com o objetivo de ser executada em eletrônicos de baixo consumo. Inicialmente, o projeto era chamado de Green Project (LUCKOW; MELO, 2010). Um dos requisitos para esse tipo de software é ter código compacto e de arquitetura neutra. Portanto, Java foi concebida para ser uma linguagem simples (não há necessidade de fazer tratamentos de ponteiros ou gerenciamento de memória) orientada a objetos e independente de sistemas operacionais. Ela acompanha uma plataforma denominada *Java Virtual Machine* (JVM), que executa o código-fonte em *bytecodes* de forma interpretada, tornando o código independente de plataforma de execução. Por fim, o Java contém um conjunto de bibliotecas que fornecem grande parte da funcionalidade básica da linguagem, incluindo rotinas de acesso à rede e criação de interface gráfica.

4.3.1.2 Spring Framework

O Spring é um projeto de código fonte aberto para a plataforma Java, criado por Rod Johnson em 2004 (JOHNSON *et al.*, 2004). Atualmente, é mantido pela Pivotal, data do referido trabalho, (FRAMEWORK, 2022b). O projeto Spring engloba um grande ecossistema, com diversos outros *frameworks* ou interfaces de configurações iniciais. Ele utiliza como base para

essas interfaces o padrão de inversão de dependência, o qual deixa sob responsabilidade do contêiner da aplicação, instanciar e destruir os objetos da aplicação, bem como a injeção de dependência, quando o próprio contêiner da aplicação entrega os objetos necessários para a lógica do negócio com o mínimo de acoplamento possível.

O ecossistema Spring, entre muitos projetos fornecidos, permite a construção de aplicações web no modo tradicional, ou seja, aplicações de múltiplas páginas com geração do HTML no lado do servidor. Para tal, ele oferece o framework Spring MVC. Por facilidade, este mesmo framework pode ser utilizado para a implementação de uma aplicação RESTful, na qual ela recebe e responde com dados no formato JSON. Além deste projeto bastante popular, vale salientar o Spring Data. Este permite abstrair totalmente o acesso às particularidades SQL, oferecendo grandes facilidades para uso de ORMs (*Object Relational Mapper*), tal como a implementação do padrão *Repository* com operações mais comuns implementadas e possibilidade de construir consultas em tabela com a sintaxe de simples métodos Java declarados no *Repository*.

O Spring Boot é um projeto da Spring que facilita o processo de configuração e publicação de aplicações. A intenção é ter um projeto em execução o mais rápido possível e sem complicação.

No Spring Boot, por meio de algumas convenções, basta ao desenvolvedor declarar os módulos desejados (i.e. Web, Template, Persistência, Segurança) no arquivo pom.xml (arquivo utilizado pelo Apache Maven para declaração de dependências) e a infraestrutura do Spring Boot se encarrega de configurar através das dependências starters. Basicamente, as dependências *starters* são grupos de outras dependências. Inclusive, por conta deste grupo representadas pelos *starters*, o arquivo pom.xml fica mais organizado. Apesar do Spring Boot entregar um projeto pré-configurado por meio das convenções adotadas, ainda é possível criar configurações customizadas. O maior benefício do Spring Boot é que ele deixa os desenvolvedores mais livres para focar nas regras de negócio da aplicação.

4.3.2 Camada de Apresentação

No modelo de três camadas, a *tier* de apresentação será responsável pela interação entre o usuário e o sistema. Nesta subseção serão expostas as tecnologias utilizadas no desenvolvimento da aplicação Xenon no lado cliente. Neste sentido, as tecnologias usadas para desenvolver a aplicação SPA serão apresentadas. Mais precisamente, serão apresentadas as linguagens de marcação HTML, CSS e o Framework Bootstrap responsáveis pela estrutura e apresentação de uma página web; também serão apresentadas as linguagens JavaScript responsáveis pelo comportamento da aplicação e na sequência, será apresentado o framework VueJS que simplifica o desenvolvimento de uma Aplicação de Página Única, ou em inglês, *Single Page Application* (SPA). Por fim, também é apresentada a linguagem de template Thymeleaf,

usada em aplicações de múltiplas páginas, a qual foi utilizada em uma versão mais inicial do Xenon.

A camada de apresentação da aplicação Xenon foi desenvolvida como uma SPA. Apesar de o nome permitir a dedução, isso não significa que a aplicação tenha apenas uma única página. Trata-se de uma forma assíncrona e dinâmica de carregar os componentes web para renderização em navegadores web. A camada de apresentação recebe um ou mais componentes por demanda da camada de lógica de domínio, sendo um componente formado por um conjunto de código referente a estrutura, aparência e comportamento de uma parte da página a ser apresentada. Os componentes são organizados ou substituídos a fim de formar a página a ser apresentada ao usuário do navegador web.

Alguns benefícios da abordagem de SPA é permitir a possibilidade de melhoramento de desempenho da aplicação ao deslocar todo o esforço para desenhar a visualização para o lado cliente e permitir um tráfego de dados mais leve entre cliente e servidor. Por outro lado, aplicações que precisem de **SEO** podem ter problemas se forem desenvolvidas com SPA. Atualmente, a maioria dos indexadores, conseguem entender aplicações SPA. Porém, é um trabalho adicional frente ao modo tradicional de renderização no modelo de múltiplas páginas no lado servidor (também chamado de *multi-page application*). Isto acontece porque a página não é disponibilizada pelo servidor, sendo desenhada completamente no lado cliente, no navegador do usuário. Por isso, os mecanismos de busca podem encontrar dificuldades para indexar o conteúdo das páginas, tendo em vista que eles não podem indexar conteúdo gerado do lado do cliente.

4.3.2.1 HTML, CSS e Framework Bootstrap

HTML (*Hypertext Markup Language*) e CSS (*Cascading Style Sheets*) são tecnologias para a construção de páginas para Web. O HTML permite construir a estrutura e apresentar o conteúdo de páginas Web por meio de uma linguagem declarativa de marcação.

Criado por Tim Berners-Lee, físico britânico, com proposito de auxiliar a disseminar conteúdo cientifico entre seus pares, que à época a solução obteve compatibilidade com a então crescente internet publica, ganhando assim atenção mundial, na década de 90 a linguagem de marcação foi definida em especificação, inspirada na proposta original de Tim Berners-Lee e então publicada em 1993 na IETF, *Internet Engineering Task Force* (WHATWG, 2021). Com o HTML é possível, como define o (W3C, 2016):

1. Construir documentos com paragrafo, tabelas, fotos, etc;
2. Navegar entre documentos através de links de hipertextos;
3. Enviar informações para aplicações remotas;

Por outro lado, a linguagem de estilo CSS, foi proposto pela primeira vez em Outubro de 1994, por Hakon Lie, que queria facilitar a construção de sites, em seu entendimento à época

era muito complexa. O criador exemplificava que era necessário muitas linhas de código para construir coisas simples, como criar uma tabela. (PACíEVITCH, 2021)

Em 1995 o CSS1 foi desenvolvido pela W3C, um grupo de empresas do ramo da informática. A linguagem de estilos ganhou muito destaque entre 1997 e 1999. Para facilitar ainda mais a criação de layouts, a W3C (*World Wide Web Consortium*) criou o CSS, colocando a disposição dos desenvolvedores. permite deixar a página Web mais amigável ao usuário humano, estilizando a página Web com cores, espaçamentos, fontes, ou seja, permitindo tratar da aparência da página. Geralmente, o CSS também é usado para a construção de *layouts* responsivos a fim de adaptar a apresentação da página para diferentes tamanhos de telas, resoluções e dispositivos (W3C, 2016).

Por sua vez, o Bootstrap é um conjunto de componentes CSS de código aberto desenvolvido para o site Twitter, chamado de *Twitter Blueprint to Bootstrap* por Mark Otto e Jacó Thornton, com o propósito de incentivar a consistência através de ferramentas internas. Antes de Bootstrap, várias bibliotecas eram utilizados para compor as interfaces, causando inconsistências e um elevado peso de manutenção, de acordo com o desenvolvedor (OTTO, 2011). No momento da publicação do referido trabalho, o Bootstrap encontra-se na versão 4, com diversos módulos, como estilos, tipografias, cores, e componentes prontos para serem utilizados na construção de páginas web. Com isso, o uso do framework facilita o trabalho do desenvolvedor, com uma linguagem padronizada diminuindo a carga de manutenção.

4.3.2.2 JavaScript, TypeScript e VueJS

O JavaScript foi criado no intuito de “tornar as páginas da web vivas”. Como os recursos em HTML e CSS são estáticos, com o JavaScript é possível manipular a estrutura, conteúdo e a aparência de uma página de forma dinâmica. O JavaScript, como o nome já se refere, trata-se de uma linguagem de *script*. Por natureza, é uma linguagem interpretada, e mais recentemente, por conta do avanço dos motores dos navegadores, pode ser considerada uma linguagem compilada em tempo de execução (*just-in-time*). O JavaScript pode ser utilizado tanto na *tier* de apresentação como também na *tier* de lógica de domínio, ou seja, pode ser executado em qualquer dispositivo que possua um interpretador, comumente chamado de máquina virtual ou motor JavaScript.

Uma das características principais do JavaScript que causa certa estranheza é a sua tipagem dinâmica. Define-se como tipagem o mecanismo de uma linguagem para definir o tipo do dado na declaração da variável, em detrimento de outras linguagens fortemente tipadas, como o Java. Por conta de alguns aspectos da linguagem, para aplicação em projetos mais complexos, podem ser usadas as linguagens ditas ”supersets”, sendo o TypeScript um exemplo mais popular.

O TypeScript é uma linguagem de código-aberto desenvolvida pela empresa Microsoft. O Typescript é uma linguagem que apresenta aderência às estruturas sintáticas das versões mais

recentes da ECMAScript e apresenta um compilador para converter o código para converter para diferentes versões do JavaScript, este processo é denominado transpilação. TypeScript é uma linguagem fortemente tipada e oferece outras características, tal como o emprego de interfaces, enumeradores, *generics* e *namespaces*. Um outro recurso tão importante é a estrutura de anotações, a qual proporciona construção de mecanismos como decoradores, que em tempo de execução, adicionam comportamentos ao *script* de forma dinâmica.(TYPESCRIPTLANG.ORG, 2021).

Por fim, o VueJS é um *Framework* para construção de aplicações Web, para *tier* de apresentação. Ele é de código-aberto publicado em 2014 por Evan You. A sua construção foi motivada pela dificuldades em dar manutenção em grandes aplicações Web construídas com JavaScript. O VueJS é utilizado para a construção de SPA modulares, isto é, cada parte da página Web pode ser dividida em módulos e então ser montada de forma dinâmica. A renderização dos dados é feita baseada em uma virtual DOM que é atualizada apenas quando os dados de um componentes são alterados, aumentando o desempenho e descartando atualizações desnecessárias (VUEJS.ORG, 2014).

4.3.2.3 Thymeleaf

O Thymeleaf é uma linguagem de template de código-aberto desenvolvida por Daniel Fernandez para integração com projetos do ecossistema Spring. Geralmente, o Thymeleaf é usado em aplicações Java construídas no modelo de múltiplas páginas para construção do HTML no lado servidor (TEAM, 2013). Ela tem o mesmo propósito do JSP(*JavaServer Page*) quando usado com o JSTL (*Java Server Pages Standard Template Library*), porém, diferindo em termos de sintaxe e capacidades.

4.3.3 Camada de Dados

Nesta subseção será observado as tecnologias utilizadas na *tier* de dados para o desenvolvimento da aplicação dissertada neste referido trabalho. Em consoante, é apresentado o Redis e o PostgresSQL.

O Redis (*Remote Dictionary Server*) é um *store* de código aberto com a estrutura de armazenamento de dados em memória criado em 2009. Ele possui a característica de flexibilidade, isto significa que ele tem suporte a muitos tipos de dados, listas, textos e conjuntos. Ele é muito utilizado para funcionalidades de cache e armazenamento de mensagens com vida útil pré-programada. Por ser simples, utiliza-se da abordagem de chave e valor, onde a chave é um identificador para o dado, e o valor é o dado propriamente dito.

O Redis começou quando Salvatore Sanfilippo, estava tentando melhorar a escalabilidade de sua *startup* italiana, desenvolvendo um analisador de log da web em tempo real. Com as dificuldades encontradas no dimensionamento de algumas cargas de trabalho usando siste-

mas de banco de dados tradicionais, a solução foi proposta, da qual seria o Redis (SANFILIPPO, 2020). O Redis costuma ser chamado de servidor de estruturas de dados. O que isso significa que o Redis fornece acesso a estruturas de dados mutáveis por meio de um conjunto de comandos, que são enviados usando um modelo servidor-cliente com soquetes TCP e um protocolo simples. Assim, diferentes processos podem consultar e modificar as mesmas estruturas de dados de maneira compartilhada.

O PostgreSQL, comumente conhecido como Postgres, é um Sistema de Gerenciamento de Banco de Dados ou *Relational Database Management System* (RDBMS) de código aberto. Ele suporta a linguagem SQL e respeita os princípios ACID, garantindo atomicidade, consistência, isolamento e durabilidade. Essas propriedades garante a integridade dos dados, não só como também, a qualidade do serviços disponibilizado pelo RDBMS, como explica Milani (2008). Por ser *multithread*, o processo é executado em múltiplos núcleos do processador o Postgres gerencia as conexões paralelas para que os dados não sejam comprometidos, como explica Milani (2008).

4.4 GESTÃO ÁGIL DE PROJETOS

O framework de gestão de projetos ágeis Scrum foi utilizado para orientar o desenvolvimento do projeto Xenon. O Scrum é uma ferramenta conceitual usada em metodologias ágeis.

> "O Manifesto Ágil reconhece que a utilização de processos, ferramentas, documentação, contratos e planos pode ser importante para o sucesso do projeto, mas são ainda mais importantes os chamados valores Ágeis: os indivíduos e interações entre eles, software (ou produto) em funcionamento, colaboração com o cliente e responder a mudanças" (SABBAGH, 2013)

O Scrum segue as diretrizes do Manifesto Ágil cunhado em 2001 na reunião de fevereiro no estado de Utah nos Estados Unidos da América. Dentro do desenvolvimento, o Scrum define alguns papéis: o *Product Owner*, o *Scrum Master* e o Time de desenvolvimento.

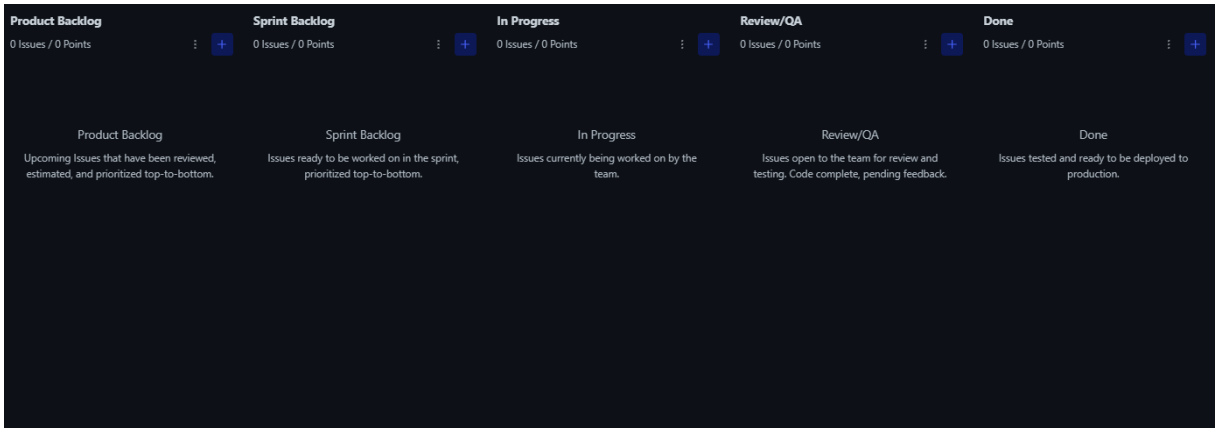
O *Product Owner* tem como responsabilidade descrever o produto a ser construído. Ele também tem o encargo de aumentar o valor do produto, pois ele deve traduzir o problema do cliente em suas reais necessidades em funcionalidades. Por sua vez, o *Scrum Master* tem como responsabilidade auxiliar o time de desenvolvimento, liderando, treinando para manter as metas alcançáveis. Por fim, o time de desenvolvimento é responsável por criar as funcionalidades do sistema ou produto em tempo hábil.

Um projeto Scrum é desmembrado em períodos menores de tempo denominados *Sprints*. Conforme SUTHERLAND (2013), um Sprint tem como objetivo criar um espaço de tempo chamado de *time-boxed* que em geral ocorre com duração de 7 ou até 30 dias para a implementação de um conjunto de funcionalidades.

Para compor o conjunto de funcionalidades, o *Product Owner* compõe o *Product Backlog* por meio da descrição de funcionalidades em forma de histórias do usuário para formar o escopo

do sistema. Geralmente, o *Product Backlog* é organizado em um quadro onde são dispostas seus itens, na Figura 6 pode ser observado as colunas Product Backlog onde ficar as histórias de usuário, na coluna Sprint Backlog as histórias que será implementadas na *Sprint* atual, já a coluna In Progress define as histórias em execução, Review/QA esta coluna define as história que foram implementadas está aguardando o crivo do time de desenvolvimento, por fim na coluna Done ficaram todas as histórias concluídas.

Figura 6 – Product Backlog

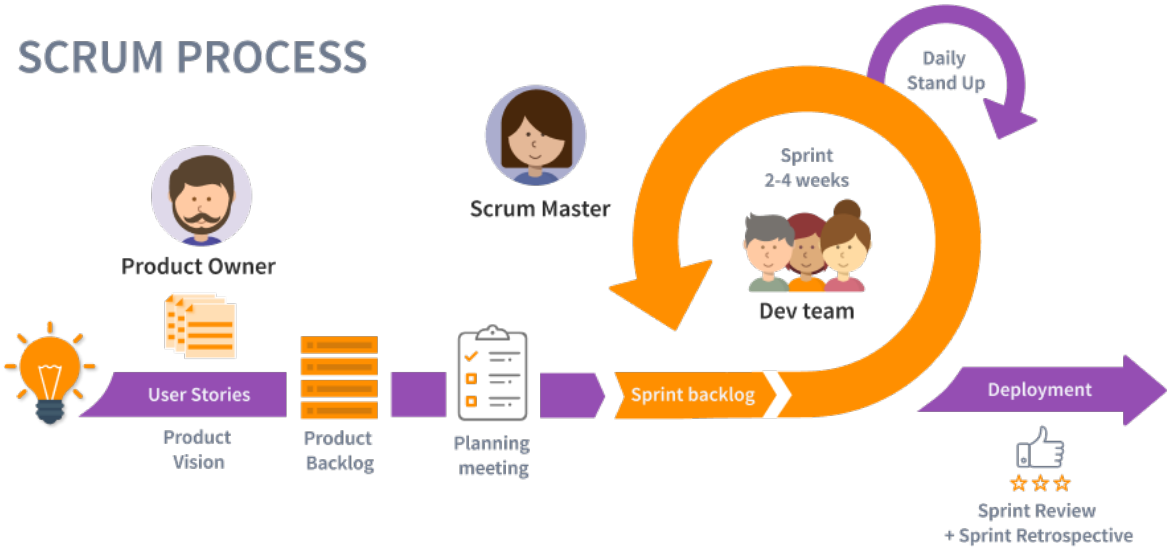


Fonte: autor.

A partir do *Product Backlog* são geradas por demanda, as *Sprints Backlog*. Estas contém as funcionalidades a serem implementadas durante um *Sprint*, podendo serem divididas em tarefas a serem executadas pelo time de desenvolvimento. Basicamente, o *Product Backlog* é uma lista inicial de funcionalidades desejadas pelo cliente, mas não necessariamente consiste em todas as funcionalidades do projeto. Em síntese, são as funcionalidades desejadas do produto. Geralmente, o *Product Backlog* cresce à medida que se aprende mais sobre o produto. Este será a peça que circula por todas as fases do projeto, podendo haver a adição de novas histórias, alteração e remoção de outras a cada final de uma *Sprint*, quando o time de desenvolvimento entrega uma parte do sistema para avaliação do *Product Owner*. Nesta reunião, o *Product Owner* recebe a parte do sistema, podendo pedir modificações, levantar novas histórias ou ainda mudar prioridades de histórias no *Product Backlog*. Este fluxo pode ser melhor explorado na Figura 7.

Como pode ser observado na Figura 7 o *Product Owner* tem uma visão geral do projeto, o produto, da qual em conjunto com o time de desenvolvimento construí o *Product Backlog*, a partir das histórias criadas, então e planejado as histórias as *Sprints Backlog* a cada novo ciclo. Também pode ser observado na Figura 7 que o time de desenvolvimento tem reuniões diárias, nestas reuniões e feito apontamentos sobre o desenvolvimento, dentre outras coisas, por exemplo, dificuldades e bloqueios que podem impactar o desenvolvimento. Ao final da *Sprint* e entregue o artefato, este ciclo será melhor explicado, posteriormente na seção 5.1.

Figura 7 – Quadro do Scrum



Fonte: Framework (2022a).

4.5 FERRAMENTAS DE DESENVOLVIMENTO

Nesta seção serão demonstrados as ferramentas que foram utilizadas no desenvolvimento da aplicação Xenon. Na subseção 4.5.1 são apresentadas as ferramentas de versionamento de código-fonte Git e GitHub, na subseção 4.5.2 é apresentada a ferramenta ZenHub, na subseção 4.5.3 é apresentada a ferramenta de integração contínua Travis-CI e por fim, na subseção 4.5.4 é apresentada a ferramenta codecov.

4.5.1 Git, Github e GitFlow

Criado pelo engenheiro de software Linus Torvalds, conhecido por ter desenvolvido, também, o núcleo Linux, o Git é um Sistema de Controle de Versões Distribuído. A sua principal funcionalidade é a guarda das alterações em arquivos, tal como códigos fonte de projetos, podendo assim intercambiar entre diferentes versões do mesmo código no tempo. O Git é muito útil para pequenos e grandes projetos, com muitos ou poucos desenvolvedores, pois com ele é possível controlar não somente as versões do código, como também quais códigos fontes serão integrados à aplicação em poucos comandos. (GIT, 2005). Para manter o código fonte acessível para equipe de desenvolvimento, fazendo com que cada integrante tenha o código atualizado a qualquer momento, se faz necessário um repositório que tenha integração com a ferramenta Git. Neste aspecto, o repositório Github.com é uma plataforma que se adequa à função. Atualmente, a plataforma GitHub é de propriedade da Microsoft e vem recebendo várias atualizações. Por meio do GitHub, por exemplo, os integrantes de uma equipe podem, avaliar o código fonte de outros integrantes, gerando um debate sobre a escrita do código, comumente chamada de code

review. Esta prática aumenta a interação com os membros de uma equipe e ajuda a manter o código fonte livre de vícios, diminuindo a chance de bugs.

O Gitflow é uma forma, um modelo de uso para gerenciar o fluxo de trabalho Git, ou seja, um conjunto de regras que de certa forma dita a forma como fazer o merge das ramificações. foi publicado pela primeira vez e popularizado por Vincent Driessen no nvie (ATLASSIAN, 2021). Neste modelo os desenvolvedores criam ramificações a partir de ramificações principais e só mesclam o código quando a funcionalidade está completa, embora o Gitflow facilita a atualização do código-fonte para os desenvolvedores, ela também podem introduzir atualizações conflitantes, pois retardam a mesclagem para a ramificação principal, deixando em certos casos o código-fonte do desenvolvedor desatualizado (ATLASSIAN, 2021).

4.5.2 ZenHub

ZenHub é uma plataforma para gerenciamento de projetos com filosofia ágil. Com uma página simples e de fácil uso, o ZenHub tem integração com Github. Isto quer dizer que com uma simples configuração, ele pode ser adicionado ao repositório do Github de um projeto. Ele contém ferramentas importantes para avaliar a saúde do projeto com relatórios e gráficos fáceis de entender. O ZenHub pode ser instalado de forma simples no repositório Github e sem a necessidade de gerenciar as tarefas, histórias, versões em outra plataforma, pois a ferramenta se integra bem no ambiente do Github (ZENHUB, 2020)

4.5.3 Travis-CI

A sigla CI significa *Continuous Integration* (integração contínua), ou seja, isto se refere a uma série de testes que pode ser escrito para garantir que uma aplicação funcione da maneira esperada através de testes unitários, integração, carga, performance e verificação do empacotamento. O Travis-CI é uma ferramenta que integra com o um repositório de código-fonte online, sendo sua principal responsabilidade manter a integridade do código-fonte, pois a cada nova atualização, o próprio Travis-CI realiza os testes nos ambientes para os quais foi configurado. A sua configuração é simples, pois faz uso de um único arquivo com a extensão .yaml.(TRAVIS, 2005)

4.5.4 Codecov

A cobertura de código é importante para a produção de software estável. Por esta razão, é importante ter uma ferramenta que avalie os testes de código-fonte. O Codecov trata-se de uma ferramenta para avaliar a cobertura de testes de código-fonte, ou seja, uma ferramenta para verificar se o código escrito apresenta falhas, ou seja, bugs.

O Codecov possui ótimas referências quando abordado para verificar códigos em Java. Além disso, ele apresenta outras três vantagens importantes: fácil integração com ferramentas de código de integração, como o Travis-CI; tem estatísticas com visualizações simples e; mensagens de e-mail informando o que mudou na cobertura de código. O Codecov lê a saída gerada por seu processo de empacotamento e produz relatórios que mostram exatamente quais caminhos de execução seu código fonte de teste perdeu. Assim, o desenvolvedor pode adicionar testes para cobrir essas regiões ou modificar a lógica dos testes existentes para fazê-los exercitar completamente a rotina que devem testar. Além disso, como o Codecov é gratuito para projetos de código aberto, existem milhares de projetos de código aberto usando Codecov também. (CODECOV, 2021)

5 ANÁLISE E PROJETO DO SISTEMA

Neste capítulo são apresentados os artefatos iniciais de análise e projeto elaborados para o desenvolvimento da aplicação Xenon. Pontualmente, na seção 5.1 são apresentados os detalhes da implementação do *Framework Scrum*; na seção 5.2 são apresentadas as histórias dos usuários na forma de Backlog do Produto e na seção 5.3 são apresentadas as histórias que compõem a primeira *Sprint Backlog* do desenvolvimento da aplicação. Em termos de decisão de projeto do sistema, a seção 5.4 apresenta a formulação da arquitetura do sistema, uma vez que ela é composta também por uma infraestrutura de hardware. Na sequência, na seção 5.6 é apresentado o resultado do projeto de modelagem do banco de dados da aplicação e por fim, na seção 5.5 são apresentados os protótipos das principais telas a serem desenvolvidas.

5.1 IMPLEMENTAÇÃO DO SCRUM

A aplicação Xenon foi desenvolvida sob as diretivas do *Framework* de gestão ágil Scrum. Primeiramente, o problema foi estudado e entendido por meio de reuniões com o *Product Owner* para levantar as histórias do usuário para compor o *Product Backlog*. Então, foi definida a primeira *Sprint*, sendo que para o corrente projeto, foi definido o intervalo de 30 dias corridos para uma *Sprint*.

A cada *Sprint*, um conjunto de funcionalidades referentes às histórias do usuário originadas do *Product Backlog* foram apresentadas para o Professor Orientador deste trabalho, sendo que este realizou o papel de *Product Owner*, por ser um funcionário da instituição e real usuário do estacionamento do campus. Em relação aos papéis de *Scrum Master* e Time de Desenvolvimento, estes foram realizados pelo próprio autor deste trabalho. Os atores mencionados anteriormente formam o *Scrum Team*.

A interação frequente do Time de Desenvolvimento com o *Product Owner* foi importante para melhorar o entendimento do problema e da solução que estava sendo entregue, pois a cada iteração das *Sprints*, o aceite das funcionalidades entregues indicaram que o processo de desenvolvimento estava traçando uma trilha correta e principalmente, a ocorrência da adição de novas histórias de usuário ou então, remoção de outras, recolocava o processo de desenvolvimento em sua trilha principal.

Este processo se deu até a última *Sprint*. Para melhor gerenciar o desenvolvimento das histórias de usuário em cada *Sprint*, estas foram subdivididas em tarefas. São exemplos de algumas tarefas: criação de testes, implementação da tarefa, testes de campo e por fim refatoração da implementação.

Para cada história concluída foi anotado seu tempo, os impedimentos quando houveram, descrevendo os motivos e o tempo para implementação. Por fim, estes dados foram de grande ajuda para compor o gráfico *Burndown* para cada *Sprint*, a fim de medir e planejar o tempo da execução das atividades.

Esta ferramenta permite melhor visibilidade do ritmo de desenvolvimento do projeto e auxilia caso novas funcionalidades e *Sprint* tenham que ser criadas.

5.2 BACKLOG DO PRODUTO

Esta seção apresenta as histórias do usuário que formam o *Product Backlog* ou Backlog do produto. A versão do *Backlog* apresentado contém as principais histórias levantadas inicialmente em reuniões com o papel de *Product Owner*. Porém, por se tratar de um projeto construído com base em um processo ágil, novas histórias foram levantadas pelo *Product Owner* durante a execução das *Sprints* e outras foram, inclusive, removidas.

Assim sendo, o complemento do *Product Backlog* será apresentado no capítulo 6, quando as *Sprints* realizadas serão apresentadas com maiores detalhes.

O *Product Backlog* inicial é composto com base nos seguintes cenários para formar o escopo inicial da solução:

1. O sistema deverá reconhecer uma placa de carro com a utilização da imagens;
2. O sistema deverá exibir dados dos motorista assim como, cadastrar, atualizar ou remover;
3. O sistema deverá cadastrar usuários com perfil de operador e administrador, bem como remover e alterar tais usuários;
4. O sistema deverá notificar quando uma placa for reconhecida e apresentando os dados do usuário, caso cadastrado anteriormente.

Com base nestes cenários mínimos, da qual representa os objetivos iniciais da solução proposta neste referido trabalho, o *Product Backlog* foi então proposto e é formado pelas histórias do usuário apresentadas nas Tabelas 1, 2, 3. Estas foram organizadas por grupos de perfil para melhor visualização. Por exemplo, as histórias relacionadas ao administrador estão agrupadas na mesma tabela. Também, as histórias estão ordenadas por grau de importância conforme definido pelo *Product Owner*. O grau de importância submete a prioridade que uma determinada história tem sobre a outra, os valores vão de 1 à 10, onde 1 é prioridade mais alta e 10 a prioridade mais baixa.

Tabela 1 – Histórias dos Motoristas.

ID	Prioridade	História
10	1	COMO Motorista QUERO me cadastrar e cadastrar meu carro.
11	2	COMO Motorista QUERO atualizar ou remover uma placa do meu carro no sistema, assim como os meus dados cadastrados no sistema.

Tabela 2 – Histórias dos Operadores.

ID	Prioridade	História
8	5	COMO Operador QUERO liberar o acesso a um motorista mediante a sinalização do sistema que ele tem autorização.
9	8	COMO operador QUERO atualizar meus dados.

Tabela 3 – Histórias dos Administradores.

ID	Prioridade	História
1	1	COMO Administrador QUERO atualizar meus dados.
2	1	COMO Administrador QUERO cadastrar novos operadores no sistema, assim como atualizar e remover um operador existente
3	3	COMO Administrador QUERO adicionar autorização de administrador a um operador já cadastrado, como também remover a autorização
4	4	COMO Administrador QUERO adicionar um novo motorista ao sistema, como atualizar seus dados ou removê-lo do cadastro.
5	5	COMO Administrador QUERO adicionar uma única placa de carro a um motorista já cadastrado.
6	6	COMO Administrador QUERO bloquear o acesso de um motorista já cadastrado, temporariamente, assim como desbloqueá-lo, assim como ver a lista dos bloqueados.
7	7	COMO Administrador do sistema QUERO ver a lista de motorista que teve acesso no dia da minha consulta.

Ademais, para cada história foi definido um identificador único, denominado ID, ou seja, um simples número sequencial utilizado para referenciá-las. Este mecanismo facilita o referenciamento da história para o *Scrum Team* e também para o texto do corrente trabalho.

Porém, além das histórias levantadas inicialmente para definir o escopo do problema, com os avanços das Sprints, quando se vai aumentando o conhecimento sobre o domínio da aplicação, novas histórias foram levantadas. Estas histórias estão apresentadas na Tabela 4.

É importante mencionar que a remoção, criação ou até mesmo modificações nas histórias foi evolutiva ao passo que o conhecimento do problema crescia e por consequência da solução era aprimorada, gerando novas funcionalidades, sendo assim a modificação no *Backlog*, este processo se deu até o fim do desenvolvimento. Os motivos para a criação de tais histórias adicionais são apresentados no capitulo 6, quando são apresentadas cada *Sprint* realizadas e com isso, a justificativa da criação, remoção ou alteração de histórias pelo time de desenvolvimento em conjunto com o *Product Owner*.

5.3 PRIMEIRA SPRINT - SPRINT BACKLOG

As histórias do *Product Backlog* foram subdivididas em tarefas para implementação dentro de cada *Sprint*, sendo que cada tarefa é uma divisão da funcionalidade descrita numa história de usuário. Após feita esta divisão, o desenvolvimento da história é iniciado. Após a tarefa ser concluída, os testes de campo são executados, quando necessário. A execução dos testes, in-

Tabela 4 – Histórias.

ID	Prioridade	História
12	1	COMO Administrador QUERO poder observar quantidade de carros cadastros do usuário.
13	1	COMO Administrador QUERO poder verificar os acessos do do usuário e seus carros cadastrados no estacionamento.
14	1	COMO Administrador QUERO poder cadastrar mais de uma portaria para ser gerenciada pelo Sistema Xenon.
15	1	COMO Administrador QUERO quando um erro ocorre em um reconhecimento, para seja salvo para poder ser feita uma auditoria sobre o caso.
16	1	COMO Administrador QUERO poder avaliar e poder recusar ou o carro cadastrado no sistema pelo motorista.
17	10	COMO Administrador QUERO ser notificado quando um cadastro de carro foi incluído no sistema para ser avaliado.
18	5	COMO Usuário do sistema QUERO recuperar minha senha quando eu esquecer.
19	5	COMO Administrador QUERO ter as informações de quantidade de usuários cadastrados separados por tipos.
20	5	COMO Administrador QUERO ter as informações de quantidade de acessos por semana.
21	5	COMO Administrador QUERO ter acesso rápido às Estações de Trabalho e a quantidade de reconhecimentos que cada uma teve.
22	1	COMO Administrador do sistema QUERO que os cadastros de alunos sejam desativado com o aluno após a conclusão do curso.
23	1	COMO Administrador do sistema QUERO quero cadastrar um usuário motorista com e configurar um tempo para a conta ficar ativa.

cluindo os testes unitários e funcionais, ocorre em ambiente controlado para verificar que as outras funcionalidades não foram impactadas.

Por fazer uso de um *Framework* ágil de gestão de projetos, e principalmente pelo caráter mais flexível do escopo quando se usa tal abordagem, será apresentada apenas a composição da primeira *Sprint*, uma vez que as histórias que comporão as demais dependerão da entrega destas e também de decisões do *Product Owner* nas reuniões das entregas. Desta maneira, a primeira *Sprint* será composta pelas seguintes histórias: ID 1, 2, 10 e 11.

5.4 ARQUITETURA DO SISTEMA

Nesta subseção são demonstradas as aplicações que correspondem ao sistema Xenon. O desenvolvimento do sistema será dividido em duas aplicações: aplicação de reconhecimento de placas e aplicação web. Por sua vez, a aplicação web é dividida em duas partes: frontend e backend. Em suma, a aplicação de reconhecimento obterá os dados de placa dos veículos e a aplicação web consumirá estes dados para apresentação ao usuário final. Basicamente, na subseção 5.4.1 são apresentados componentes que formam a aplicação de reconhecimento de placas. Na subseção 5.4.2, são apresentados a arquitetura da aplicação web. Seguindo na subseção 5.4.3 será demonstrado a separação física. Em seguida na seção 5.5 a prototipagens da tela para a aplicação web. Por fim mas, não menos importante na seção 5.6 será apresentado a modelagem do banco de dados.

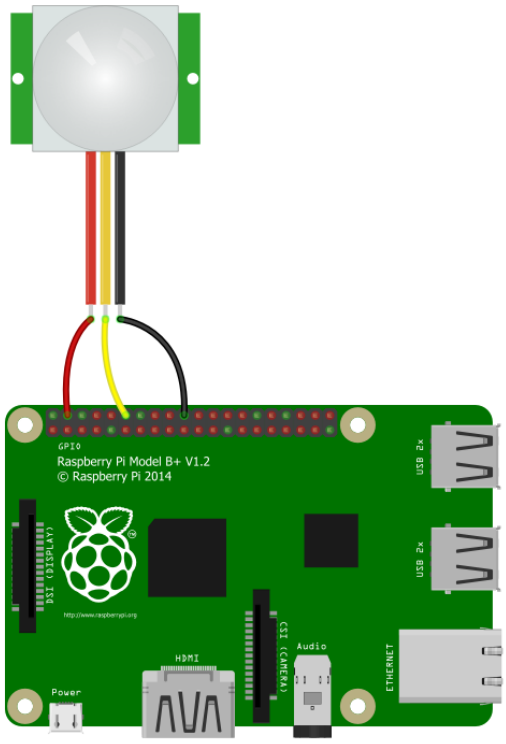
5.4.1 Aplicação de reconhecimento de placas

A aplicação de reconhecimento de placas executará na plataforma Raspberry PI, conforme explanado na subseção 2.2. Esta plataforma de hardware possui conexões (conexão GPIO para enviar um sinal a um componente externo) e componentes, tais como: Sensor PIR e o módulo de Câmera. O sensor PIR será instalado no Raspberry, para ser o gatilho para tirar a foto do carro, será jumpeando nos seguintes pinos:

- Pino 4, 5v de uma das conexões VCC;
- Pino 18, sinal de dados da conexão GPIO;
- Pino 30, de uma das conexões GND.

A Figura 8 exemplifica a instalação do sensor PIR. O sensor será ajustado para a distância mínima de 3 metros e com intervalo de 5 segundos de tempo de acionamento do sinal. É importante ressaltar que o sensor deve ser apontado para a mesma direção que o módulo da câmera estará direcionado. O Raspberry comporta uma porta específica para módulo de câmera, com suporte para conexão CSI pelo cabo flat, como mostra a Figura 2.

Figura 8 – Conexões entre Raspberry e o Sensor PIR

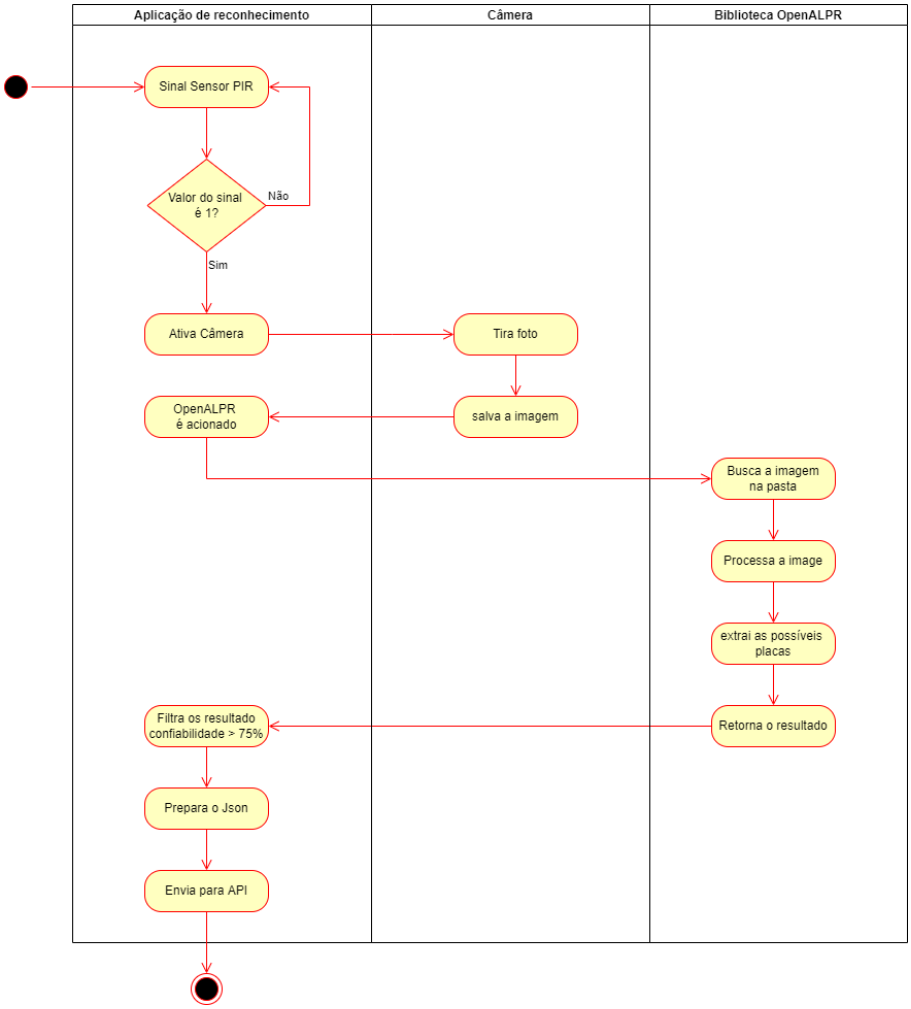


Fonte: autor.

O módulo da câmera será acionado quando o sensor PIR for ativado, tirando uma foto do momento, que será salva no Raspberry. É importante mencionar que esta foto será dinâmica, ou seja, será sempre substituída pela próxima foto, mantendo sempre a última imagem adquirida.

A aplicação de reconhecimento, a qual possui a responsabilidade de se comunicar com a API no backend da aplicação web, será explicado na subseção 5.4.2. Esta aplicação faz uso da biblioteca OpenALPR como componente auxiliar na execução do algoritmo de reconhecimento ALPR, o qual processa a imagem da placa recuperada da aplicação de reconhecimento, dando as devidas atenções ao processamento da imagem. A aplicação de reconhecimento, processará a imagem enviando para OpenALPR fazer o reconhecimento dos caracteres da placa. Uma vez retornado o resultado, a aplicação então prepara os dados dos reconhecimentos das placas e envia para a API no backend. A aplicação de reconhecimento de placas é a orquestradora dos componentes físicos supracitados. Quando a aplicação é iniciada, ela fica observando o sinal do sensor, que quando o valor for 1, emite um sinal para a câmera tirar uma foto, a qual é salva em uma pasta específica no Raspberry. Na sequência, a biblioteca OpenALPR é invocada para extrair a placa da imagem. O OpenALPR foi programado para sempre buscar imagem no diretório já definido. Então, estes resultados são filtrados de acordo com uma taxa de confiabilidade, ou seja, uma taxa maior do que 75.0% para enviar para a API no backend. Na Figura 9 é demonstrada a atividade do fluxo de reconhecimento.

Figura 9 – Diagrama de atividade da aplicação de integração



Fonte: autor.

Por fim, a aplicação de reconhecimento foi desenvolvida na linguagem JavaScript para execução no ambiente NodeJS na plataforma Raspberry. Esta aplicação também possui um endpoint para receber notificações do backend para enviar um sinal para uma porta GPIO, a qual pode ser conectada a um relé para exercer o comando de abrir uma cancela, por exemplo. Entretanto, esta função está presente no sistema, mas o escopo do projeto não contempla a instalação de outro periférico, como por exemplo, a própria cancela.

5.4.2 Aplicação Web

O Backend ou API (Application Programming Interface em inglês que em tradução livre é Interface de Programação de Aplicativos) foi construído com a utilização da linguagem Java com uso do ecossistema de frameworks Spring. Inicialmente, a camada de visão da aplicação seria realizada com a *engine* Thymeleaf, porém, na análise do sistema, decidiu-se pela implementação da camada de aplicação de forma separada do backend, utilizando para tal o framework VueJS para compor o frontend. O Frontend conterá as telas do usuário e tratará dos eventos enviando requisições para backend, além de manter uma comunicação dos reconhecimentos em tempo real, atualizando a interface do usuário a cada novo reconhecimento.

API recebe não só as requisição da aplicação de reconhecimento, como também as ações dos usuários oriundas da aplicação de interface do usuário. Este processo pode ou não invocar o banco de dados para salvar ou recuperar algum dado já processado anteriormente. Por fim, o resultado será enviado para *tier* de apresentação.

Como já mencionado, esta aplicação tem também uma interface para as funcionalidades, que receberá requisições da interface do usuário para alterar, remover e adicionar novos administradores e operadores. Ademais, ela também disponibilizará uma interface para os motoristas poderem alterar os dados de seus carros. Esta API tem como propósito dispor a *tier* de apresentação às funcionalidades que o sistema Xenon propõe para os usuários.

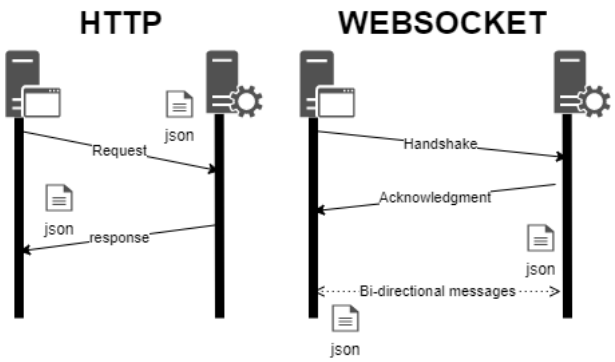
5.4.3 Separação Física

O Frontend estará instalado em um servidor executando o servidor Web configurado para servir o navegador com os arquivos JavaScript a fim de construir a interface do usuário. Por sua vez, o Backend será instalada em outro servidor utilizando servidor Web já incluso no *Framework Spring*. Por fim, o banco de dados Postgres também será separado fisicamente, ou seja, instalado em outro servidor. Neste mesmo, no servidor será instalado o Redis.

A comunicação entre o Backend e o Frontend será via protocolo HTTP com carga útil em JSON de forma assíncrona. Um outro ponto de comunicação entre ambas as partes terá um canal de comunicação assíncrona sobre o protocolo *Websocket*. Este será utilizado para

a comunicação de reconhecimento em tempo real. A Figura 10 exemplifica a diferença entre ambas comunicações.

Figura 10 – Comunicação HTTP e WebSocket

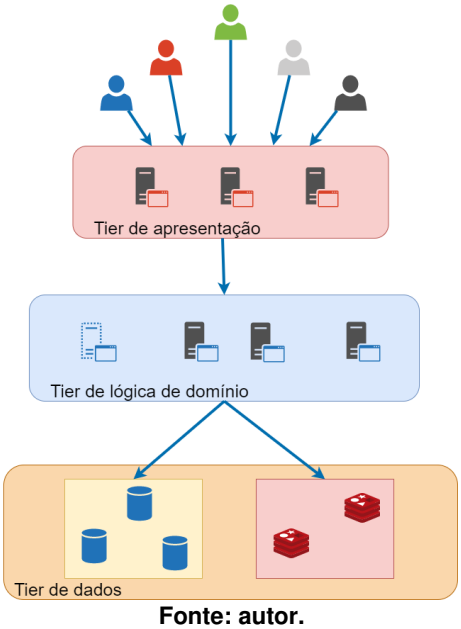


Fonte: autor.

A comunicação entre o Backend e o servidor de banco de dados ficará a cargo da API JDBC do Java. O Spring Framework fará o serviço para esta comunicação, não sendo necessário nenhuma configuração no servidor de banco de dados para a comunicação efetiva. Como já mencionado anteriormente, o servidor de mensageria e cache, o Redis, também ficará no mesmo servidor do Postgres.

Esta separação física entre as aplicações proporciona a escalabilidade horizontal, gerando um menor custo quando é necessário aumentar o poder de processamento das requisições. Assim, é possível aumentar separadamente cada *tiers* de forma independente, sem impactar nas outras aplicações. Como o inverso também é verdadeiro, é possível diminuir a quantidade de servidores de uma *tier*, gerando economia. A Figura 11 exemplifica esta dinâmica.

Figura 11 – Comunicação entre as camadas

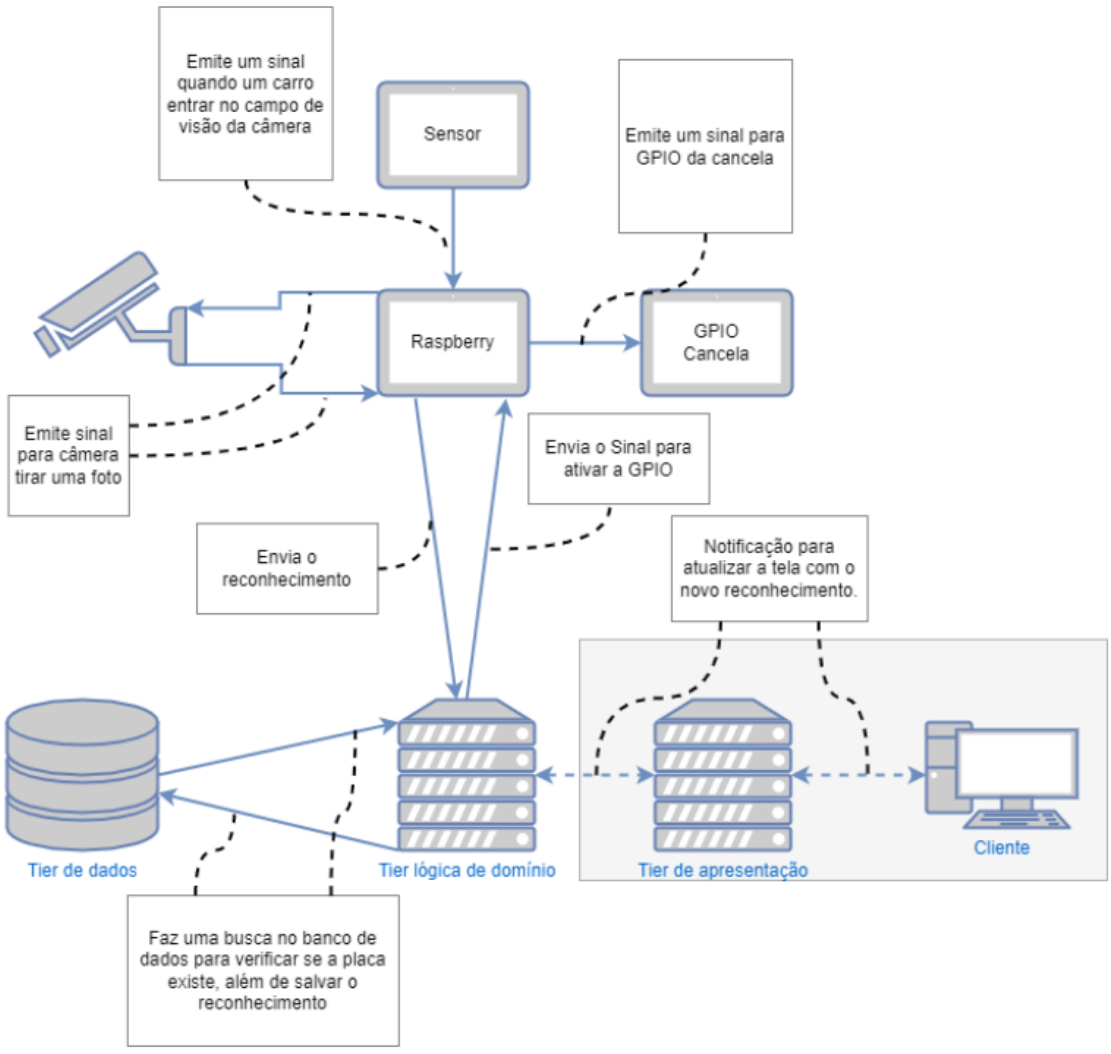


Fonte: autor.

Um outro ponto importante na arquitetura, se refere ao objeto inteligente, implementado na plataforma Raspberry. Este é responsável por fazer o reconhecimento das placas e é conectado dentro da mesma rede de computadores em que residem as aplicações mencionadas anteriormente. Nesta rede, o fluxo de comunicação dará sobre o protocolo HTTP para comunicação síncrona. O formato para trafegar os dados entre a aplicação de lógica com as aplicações de apresentação e Raspberry, será JSON.

A dinâmica de comunicação entre as aplicações é melhor exemplificada com auxílio da Figura 12. Esta ilustra uma visão global do fluxo de atividade das quais o sistema executará. O exemplo é de um reconhecimento, ou seja, quando um motorista entra no campo de visão da câmera que alimenta o sistema. É importante ressaltar que o balanceamento de carga não é objeto de estudo do referido trabalho.

Figura 12 – Comunicação das camadas



Fonte: autor.

5.5 PROTOTIPAGEM DE TELAS E FUNCIONALIDADES

A história de ID 10 da Tabela 1 refere-se ao cadastro de um usuário motorista. Este cadastro é destinado ao corpo discente do campus para submeter um pedido de acesso ao estacionamento do campus. Para esta funcionalidade, foi projetada uma tela da interface do usuário, como mostrado na Figura 13.

Seguindo com as histórias da Tabela 1, a história de ID 11 refere-se à funcionalidade para alterar ou remover os dados do usuário. O protótipo apresentado na Figura 14 refere-se a esta funcionalidade. No entanto, a mesma tela será utilizada para as histórias de ID 9 da Tabela 2, como também a história de ID 1 da Tabela 3.

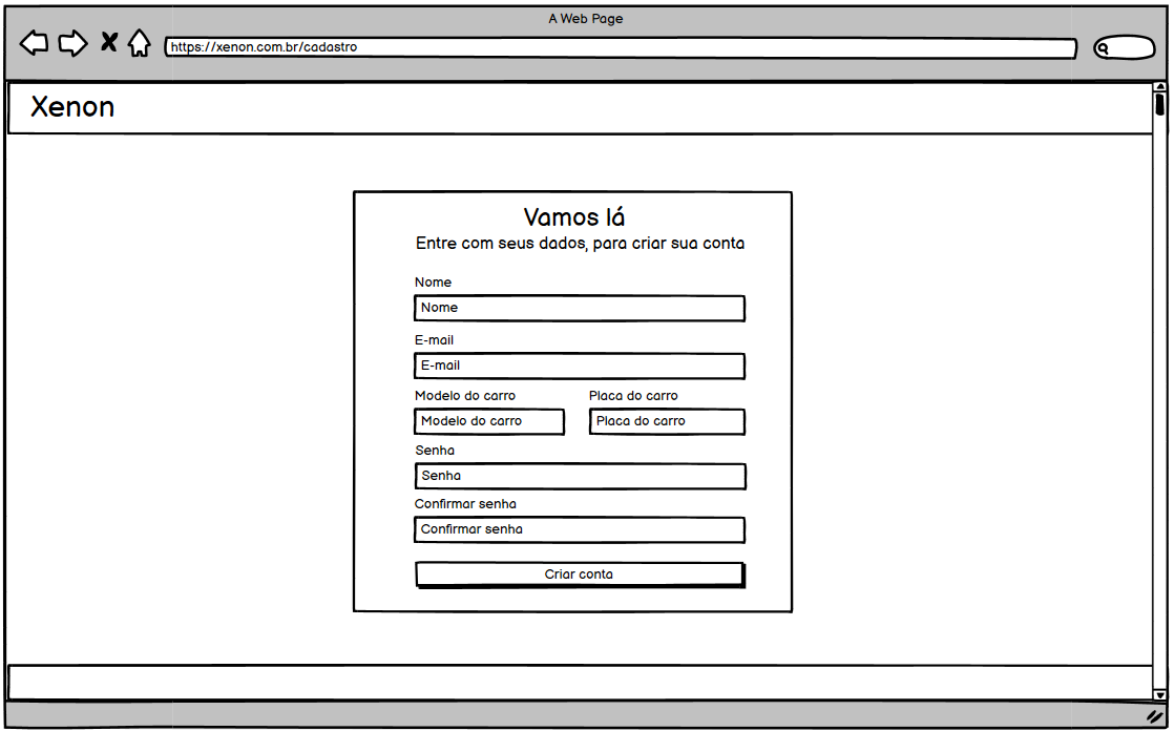
Por sua vez, o protótipo apresentado na Figura 15 refere-se à tela da funcionalidade para alterar a senha e por fim, o protótipo apresentado na Figura 16 será usado para adicionar um carro na conta.

O protótipo de tela ilustrada na Figura 17 refere-se à funcionalidade para cadastrar um usuário com perfil de Administrador, abrangendo a história de ID 2. Este mesmo protótipo também será utilizado para atualizar os dados do usuário, assim como também para implementar as histórias de ID 3, 4 e 5.

Na Figura 18 é apresentada a tela para a implementação das histórias com ID 6 e 7. Mais precisamente, a história de ID 7 remete à funcionalidade para listar todos os usuários cadastrados no sistema e a história de ID 6 refere-se à funcionalidade para bloquear e desbloquear o acesso do usuário.

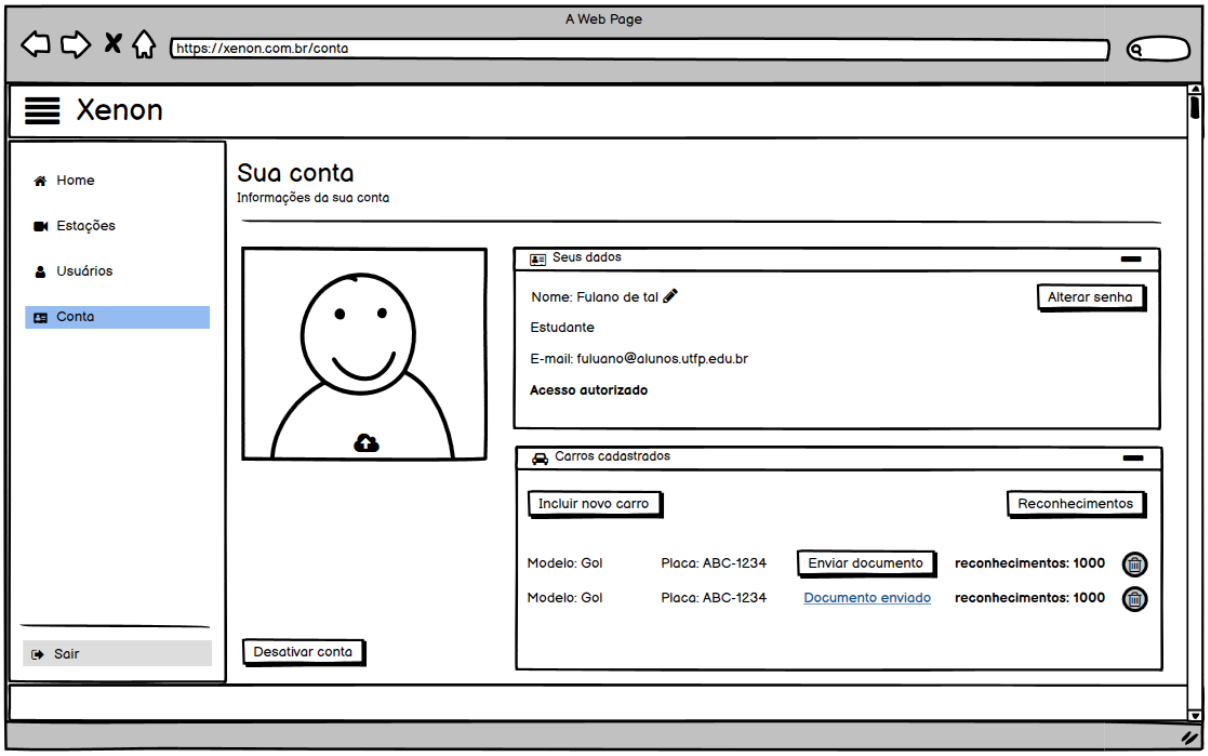
Na sequência, a história de ID 8 da Tabela 2 é extraída da funcionalidade principal do sistema referente à exibição da placa reconhecida a partir da Aplicação de Reconhecimento e liberação ou não do acesso ao veículo. O protótipo da tela é ilustrada na Figura 19.

Figura 13 – Cadastro de conta



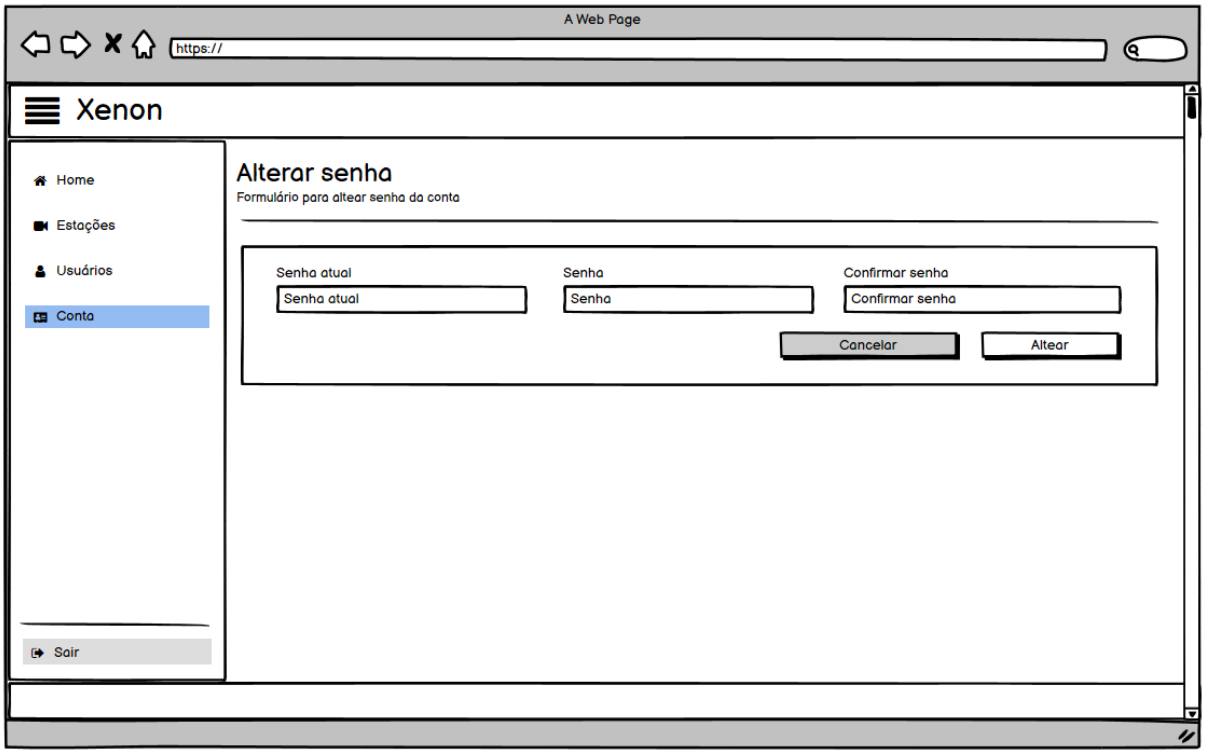
Fonte: autor.

Figura 14 – Tela para mostrar os dados da conta do usuário



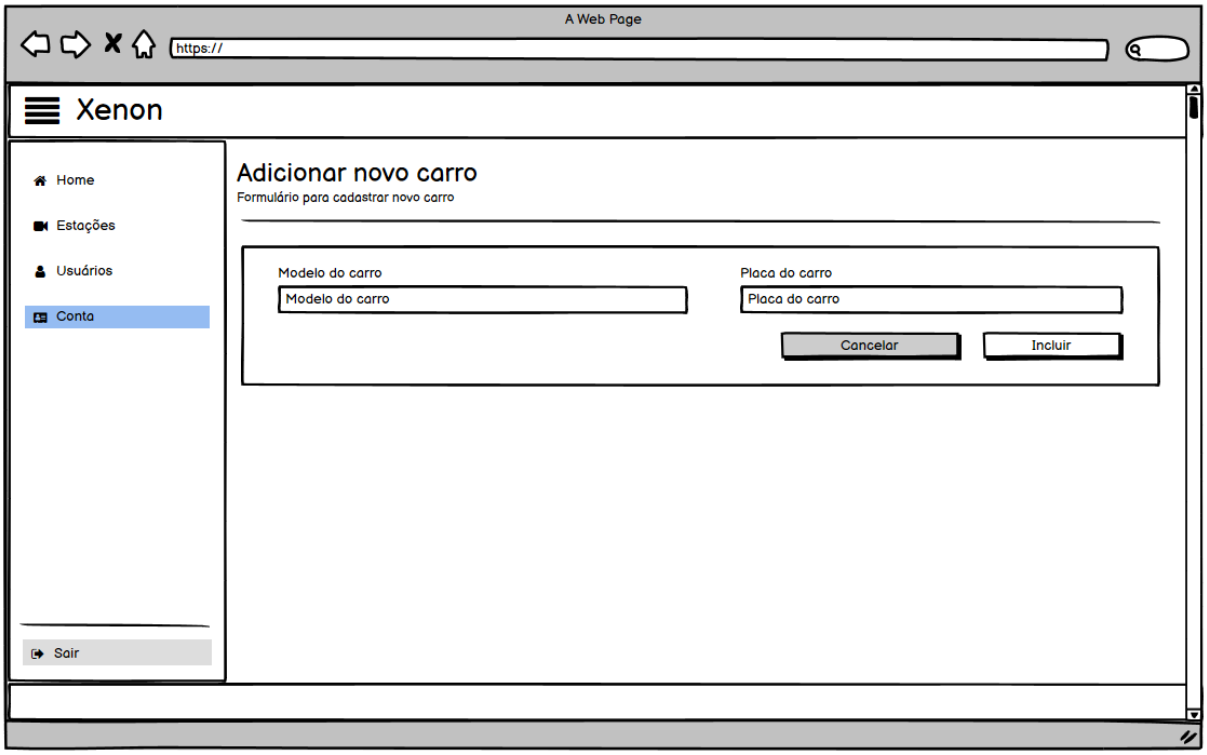
Fonte: autor.

Figura 15 – Tela para alterar a senha



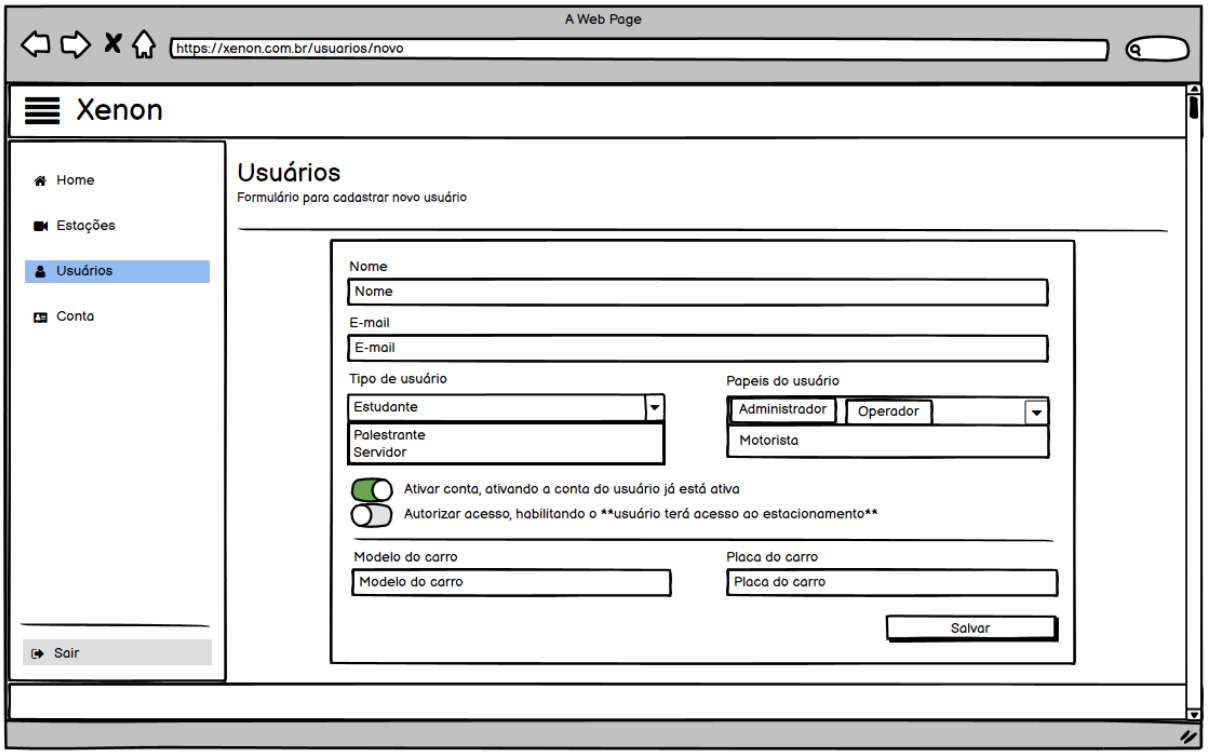
Fonte: autor.

Figura 16 – Tela para adicionar um carro



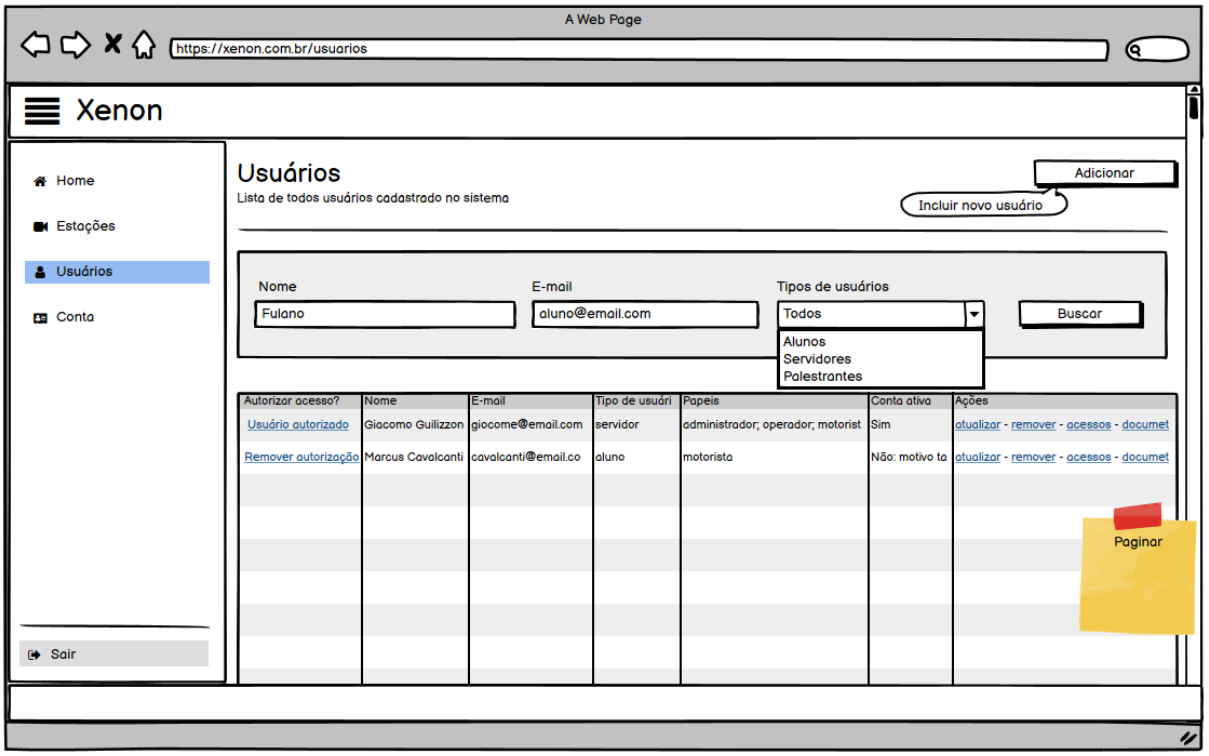
Fonte: autor.

Figura 17 – Tela para cadastrar e alterar um usuário



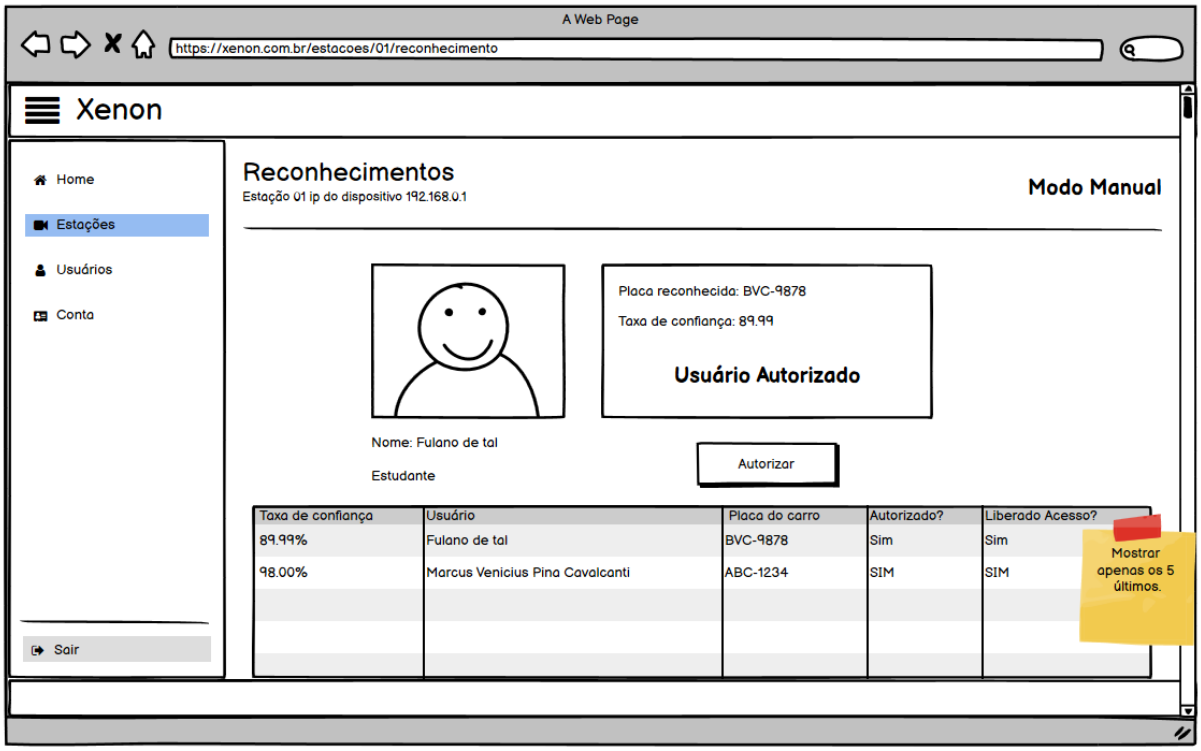
Fonte: autor.

Figura 18 – Tela de listagem de usuário



Fonte: autor.

Figura 19 – Tela de reconhecimento recebido



Fonte: autor.

5.6 PROJETO DO BANCO DE DADOS

O modelo de banco de dados principal foi concebido como um esboço inicial para a análise do sistema Xenon, antes de iniciar as Sprints e foi aperfeiçoado durante o avanço das mesmas. No decorrer da evolução do desenvolvimento das Sprints, algumas tabelas foram adicionadas, assim como outras foram removidas. Estes casos são mencionados com maiores detalhes no capitulo 6, ou seja, quais as alterações pontuais feitas a partir deste modelo de dados inicial. Na Figura 20 é apresentada a modelagem da base de dados inicial do sistema.

Ainda, vale ressaltar que a aplicação de reconhecimento de placas que executa no objeto inteligente não necessita de banco de dados propriamente dito. Para o processo de aprendizagem é necessário de apenas um arquivo de texto, que contém o *Dataset*, ou seja, os dados de informação das entradas e saídas da aprendizagem do OpenALPR, a qual foi demostrada como aprendizagem supervisionada.

Conforme a Figura 20, pode-se aferir que o sistema foi projetado para ter uma entidade chamada *AccessCard*. Esta entidade será responsável por salvaguardar as credenciais dos usuários, seja ele qualquer um dos tipos de usuário que o sistema terá, que são eles:

1. **UserOperator**: pode ser um usuário, administrador ou operador. Ou seja, variantes que depende do perfil atribuído a ele. Este usuário pode representar o usuário operador

do sistema, funcionários da empresa de segurança e componentes da comissão do estacionamento.

2. **UserDriver**: como o nome sugere, esta entidade representa o usuário motorista, ou seja, o usuário que pede acesso ao estacionamento do campus.

Também na Figura 20 pode ser observador a tabela Roles. Esta é a entidade que guarda os dados referentes aos perfis do sistema Xenon, os quais são: **ROLE_USER, ROLE_ADMIN e ROLE_OPERATOR**.

Ambas as entidades apresentadas anteriormente são sugeridas pelo *Spring Framework* que em conjunto com um módulo denominado *Spring Security* que tem como proposito separar as entidades de domínio, ou seja, entidades com valor semântico para lógica de negocio, da segurança. Esta abordagem de separar usuário das suas credencias de acesso, trás como beneficio o isolamento, pois as credencias de acesso de um determinado usuário o compõe, sendo assim o usuário não deve ter como comportamento intrínseco, isto fica a responsabilidade de saber como, quando e porque de acessar o sistema fica a cargo da entidade *AccessCard*, em outras palavras a entidade *AccessCard* dá valor a entidade *User* que não precisa daquela para existir e sim ao contrário. Outro ponto importante neste relacionamento entre as entidades, mencionada anteriormente, que como pode ser observado na tanto na Figura 20, quanto na Figura 21 é a coluna **type**, na tabela de **users**, da qual tem 3 valores pré-fixados, que são eles:

- **SERVICES**, que define que o usuário e um servidor/colaborador;
- **STUDENTS**, este valor indica que o usuário é um discente;
- **SPEAKER**, o valor dado para um usuário palestrante.

Esta coluna é utilizada para delimitar as fronteiras das perfis de usuário, um usuário *SERVICES* poderá receber todos os perfis de usuário, ao passo que para os dois restantes apenas apenas o perfil de motorista. Isto facilita a customização dos perfis de usuário, pois outras combinações podem ser incrementadas posteriormente.

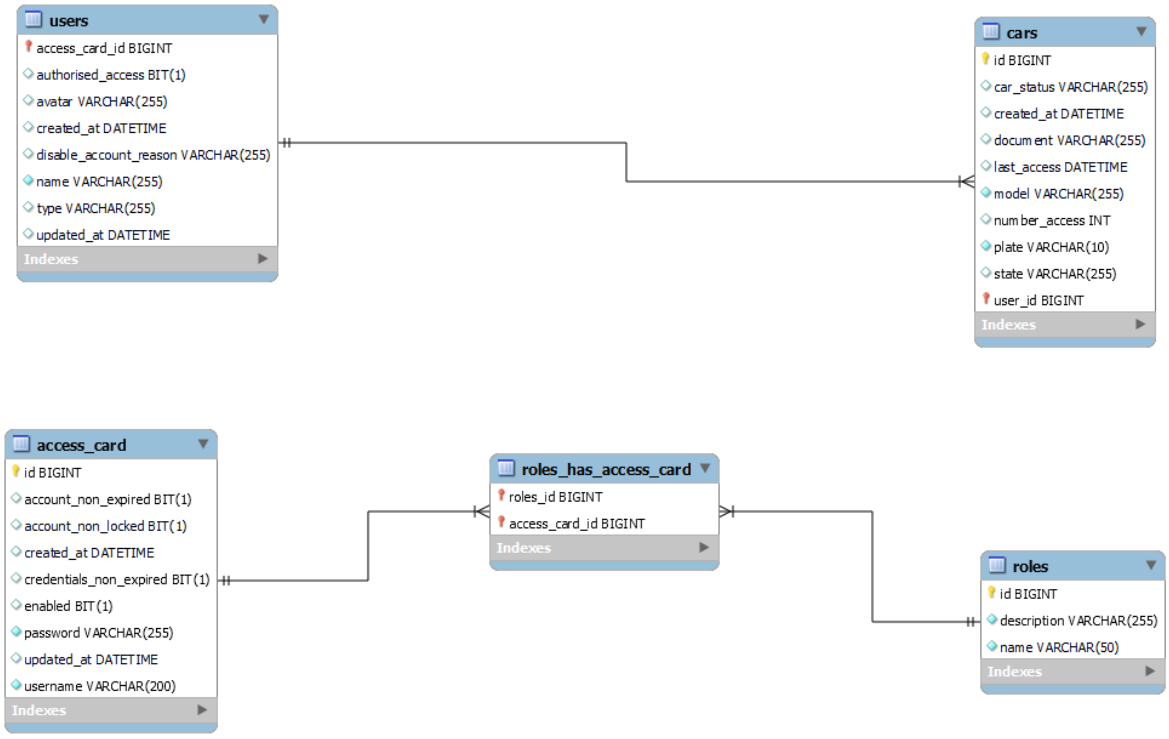
Como a modelagem do banco de dados apresentado na Figura 20 é apenas uma representação inicial da organização dos dados do sistema, na Figura 21 é apresentada a versão final do banco de dados. Durante o desenvolvimento, algumas novas funcionalidades foram criadas, sendo necessário para algumas delas a criação de novas tabelas, como por exemplo, a tabela **error_recognizer**. Esta tem a responsabilidade de guardar os dados de erro quando algum reconhecimento não for bem sucedido devido a algum erro.

Já a tabela **recognizers** foi criada para suportar operações de auditoria a fim de verificar quando e onde a placa foi identificada, assim como a confiabilidade do reconhecimento.

Por fim, foi adicionada a tabela **workstations**, na qual são guardados os dados das portarias. Inicialmente, a aplicação web iria apenas receber reconhecimentos de apenas um local. Porém, no desenvolvimento, foi observado que deveria ter mais de uma portaria, sendo

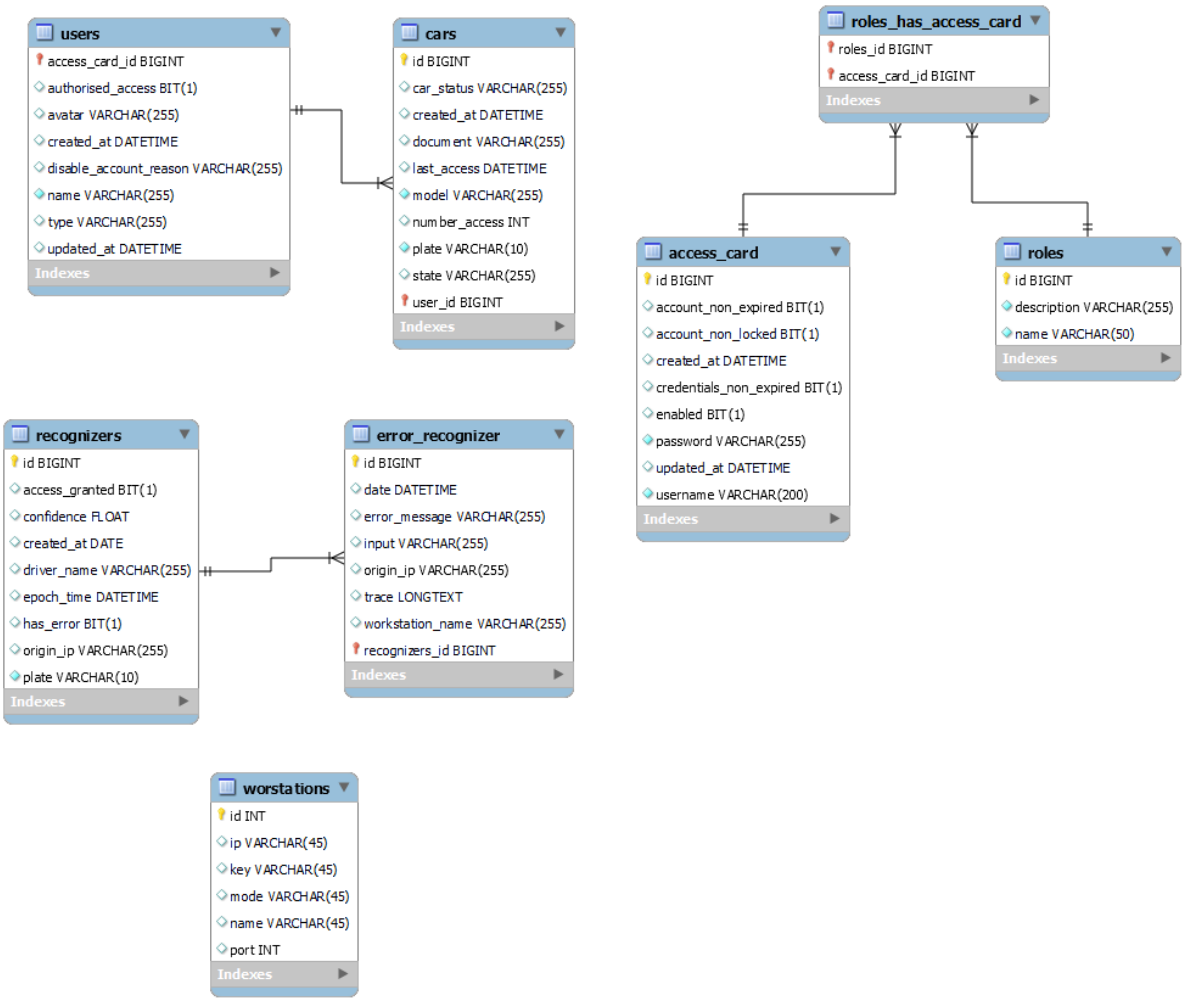
assim necessário esta tabela. O vínculo com o Raspberry ocorre utilizando as colunas **ip** para o endereço na rede e **port** para a porta de rede. Para manter a identificação, a coluna **key** é utilizada para guardar a chave de identificação que será única pra cada portaria. As novas tabelas serão melhor justificadas no capítulo 6 de desenvolvimento das Sprints.

Figura 20 – Modelagem inicial do banco de dados



Fonte: autor.

Figura 21 – Modelagem final do banco de dados



Fonte: autor.

6 DESENVOLVIMENTO DO SISTEMA

Neste capítulo serão apresentadas de forma sucinta as fases do desenvolvimento do Sistema Xenon em forma de Sprints conforme o processo Scrum. Propriamente, cada seção explana cada Sprint realizada, da primeira até a ultima *Sprint*. Basicamente, este capítulo está estruturado em forma de um relatório do histórico de desenvolvimento das histórias do usuário, da seção 6.1 até a seção 6.7, sendo que na seção 6.8 são apresentadas as atividades de configuração e treinamento da aplicação de reconhecimento que ocorreram de forma incremental e transversal a todas as Sprints. Por fim, na seção 6.9 são apresentadas as informações pertinentes ao desenvolvimento.

6.1 SPRINT 1

O desenvolvimento da primeira *Sprint* consiste na implementação das histórias de ID 1[1], 2[2], 10[3] e 11[4], conforme solicitado pelo *Product Owner*. A primeira história implementada foi a de número 2, sendo que esta é o ponto de partida para inclusão de usuários. As outras histórias nesta *Sprint* foram completadas após a estrutura inicial da história de ID 2. Para implementar o cadastro de usuário por meio de uma ação do administrador, o *Scrum Team* diluiu a história em algumas tarefas, estas são:

- **Tarefas**:

    1. Implementar a segurança, para identificar o administrador.
    2. Implementar a funcionalidade para listar usuários;
    3. Implementar a funcionalidade para cadastrar um usuário;

Conforme as tarefas supracitadas, para construir a camada de segurança foi feita a criação do banco de dados inicial, além de adicionar as tabelas de usuários, perfil de usuário e dos carros. Uma vez criada a modelagem do banco de dados, foi então implementada a camada de segurança, utilizando a estrutura do *framework* Spring. O mecanismo de autenticação foi implementado usando JWT (*Json Web Token*). Para tal, a tela de login foi implementada e ficou conforme demonstra a Figura 22.

Com a implementação da camada de segurança, foi possível implementar a funcionalidade de cadastro de usuário advinda de uma ação do administrador. Com isso, a URL do

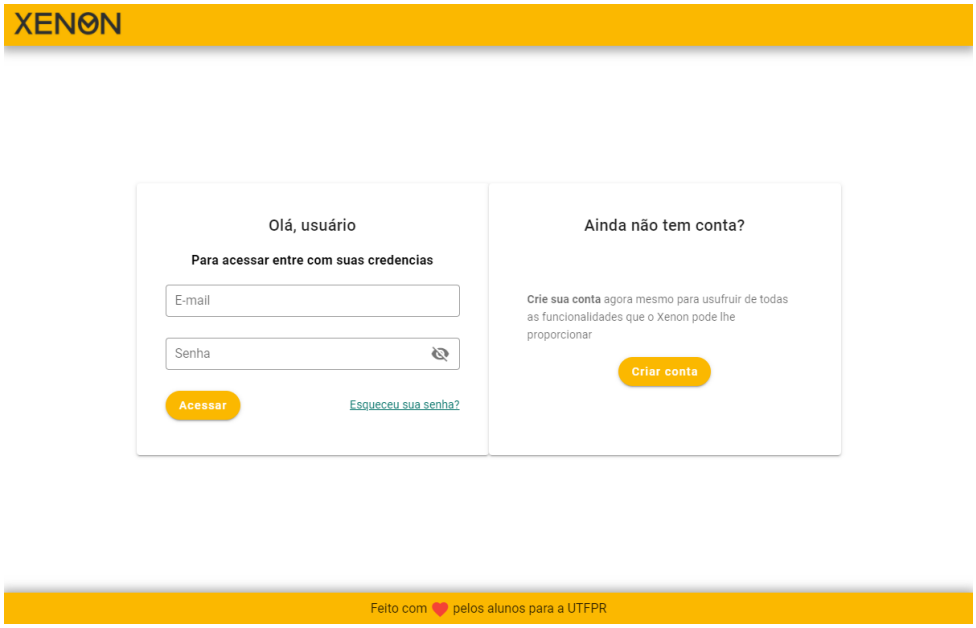
[1] Como administrador gostaria de atualizar meus dados

[2] Como administrador gostaria de cadastrar novos operadores no sistema, assim como atualizar e remover um operador existente

[3] Como motorista quero me cadastrar e cadastrar meu carro.

[4] Como motorista quero atualizar ou remover uma placa do meu carro no sistema, assim como os meus dados cadastrados no sistema

Figura 22 – Tela de login



Fonte: autor.

recurso usuário na API passou a ser protegida por autenticação e o acesso restringido ao papel de usuário administrador.

Ademais, o *Scrum Team* observou que não faria sentido um administrador incluir um carro para o usuário, dado que é necessário incluir um documento do carro. Foi observado que, caso o administrador cadastrasse um carro para o usuário que ele criou, o acesso estaria devidamente liberado mas, com dados inconsistentes, uma vez que o documento por questões de privacidade deveria ser adicionado pelo usuário. Logo, dois cenários foram levantados: o primeiro dependeria do usuário incluir no seu cadastro o documento do carro, para assim concluir o cadastro; e o outro cenário dependeria do administrador incluir no momento do cadastro.

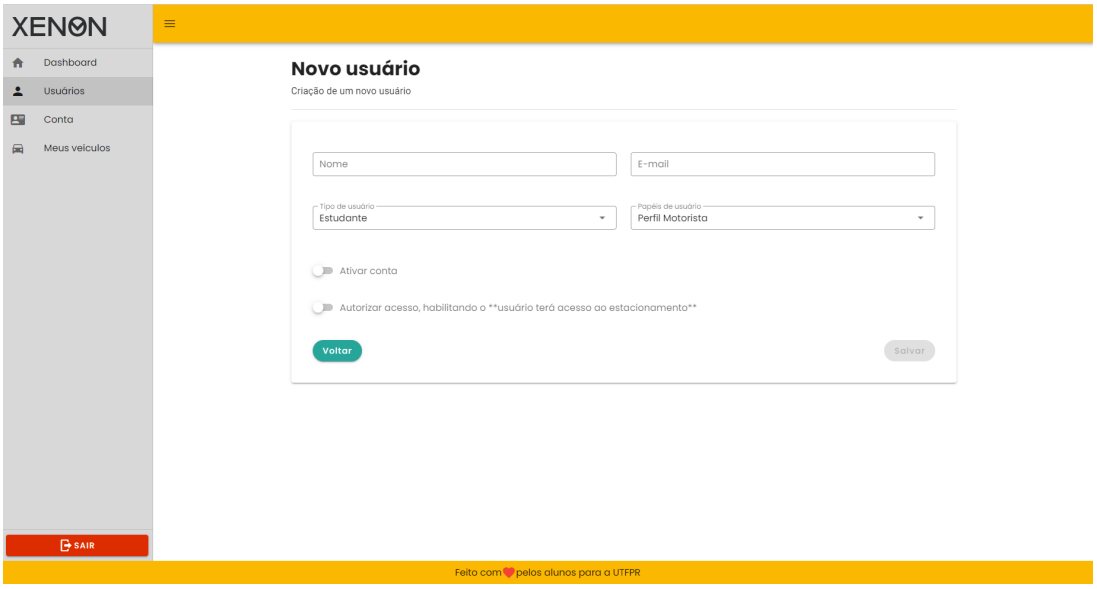
Ambas as opções tem problemas, não faria sentido o administrador criar um cadastro que para conclusão dependia da ação de outro usuário para concluí-la, o usuário motorista atualizar o cadastro do carro com o documento do veiculo, aumentando a complexabilidade da funcionalidade.

Ao passo que na segunda hipótese levantada, o usuário motorista deveria necessariamente estar presente no momento do cadastro ou então enviar uma cópia do documento por email ou outro meio de comunicação, para que o administrador pudesse incluir seus dados e os dados do carro. Nesta opção, não só aumentaria a complexabilidade da funcionalidade, como também impactaria no sentido do cadastro do carro pelo usuário motorista, pois o administrador teria que cadastrar um usuário motorista e depois incluir o cadastro do carro, para concluir o cadastro do usuário motorista, precisando preferencialmente o motorista estar presente.

Estas opções vão de encontro com o propósito da aplicação de simplificar e automatizar o acesso ao estacionamento do campus. Neste sentido, seriam passos desnecessários para o cadastro do usuário. Assim, para solucionar o problema foi removido do formulário de cadastro

da área do administrador, os campos destinados ao carro, como mostrado na Figura 23. Com isso, apenas o usuário motorista poderá adicionar ou remover um carro, deixando o fluxo de cadastro mais simples, pois o administrador fica só com responsabilidade apenas de criar a conta do usuário e verificar as informações incluídas pelo usuário motorista no que tange ao cadastro do carro. Impedindo assim inconsistência nos dados e deixando a funcionalidade de inclusão de carro mais fluída. Com isso, quando um usuário motorista mudar de carro, ele simplesmente fará a atualização no sistema.

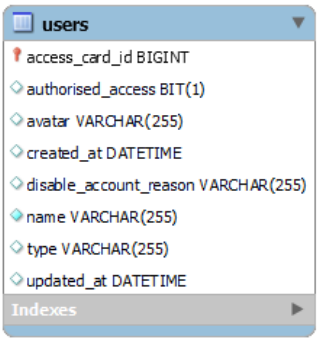
Figura 23 – Tela do formulário de usuário



Fonte: autor.

Também vale mencionar que *Scrum Team* decidiu que o papel de Operador e Administrador só e somente só será concebido para usuários que são do tipo Servidor, conforme o valor do campo **type** da tabela **users**, apresentado na Figura 24. Com isso, evita-se que alunos tenham privilégios que não se enquadram no tipo de usuário. Outro ponto que foi incluído após análise do sistema foi a remoção do valor do RA do aluno, haja visto que apenas o email com a confirmação já era mais do que o suficiente para identificar o discente.

Figura 24 – Tabela de usuário



Fonte: autor.

Seguindo para a história de ID 10, uma questão que foi definida em refinamento do *Scrum Team*, refere-se à funcionalidade para o motorista se cadastrar. Esta funcionalidade seria apenas para o corpo discente da UTFPR. Com base nesta proposta, então foi definido que o usuário discente teria que obrigatoriamente utilizar seu email institucional. Logo, foi necessário incluir um fluxo para confirmar o email do usuário estudante ativo e concluir o seu cadastro.

As tarefas criadas para a história 10, são:

- **Tarefas**:

    1. Implementar a funcionalidade para os motoristas poderem se cadastrar;
    2. Implementar o fluxo para envio de email para validar a conta;
    3. Implementar o fluxo para validar a conta de email.

O fluxo para confirmar o email institucional do discente consiste no envio de um link de confirmação para o email informado no cadastro, quando o discente confirma a veracidade do email clicando no link recebido, tal como demonstrado na Figura 25. Ao confirmar o email, a conta do usuário estará cadastrada e ativa no sistema Xenon.

Figura 25 – E-mail para ativar a conta

Fonte: autor.

Após a conclusão da história 10, foi a vez da histórias de ID 1 e 11. Desta maneira, as tarefas para estas histórias são:

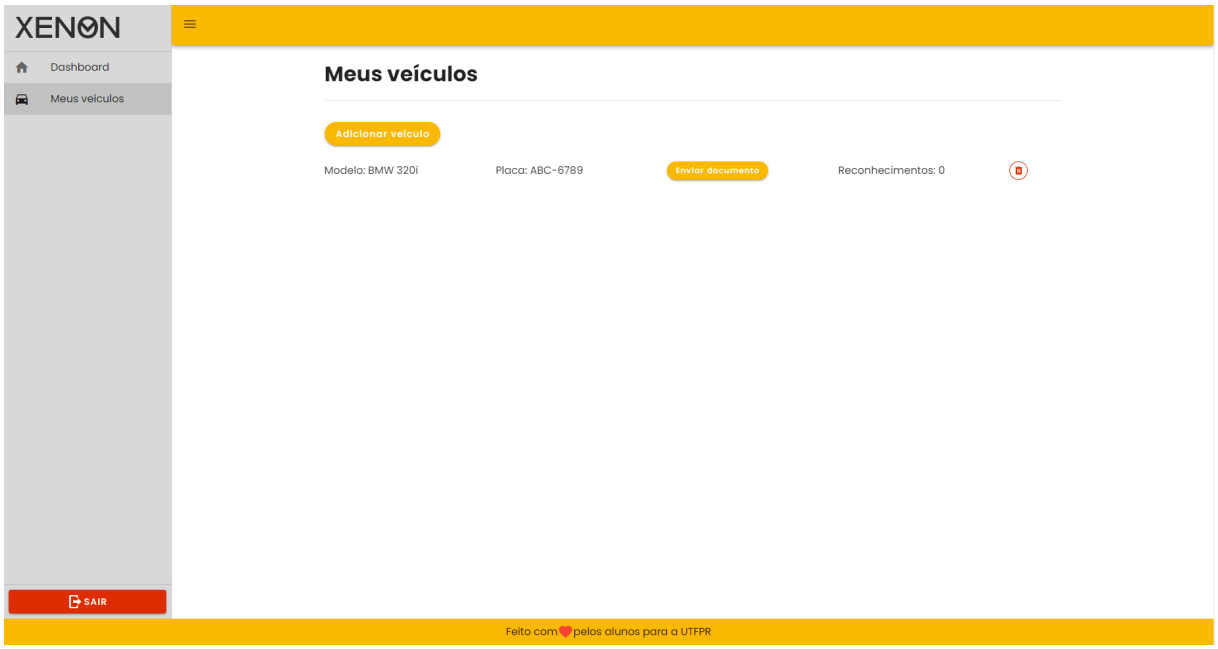
- **Tarefas**:

    1. Implementar a funcionalidade para ver os dados da conta;
    2. Implementar a funcionalidade para o usuário incluir carros;

O *Product Owner* decidiu que a funcionalidade para cadastrar um carro e listar os carros cadastrados no sistema deveria ser separada, sendo necessário criar uma tela para esta funcionalidade, que anteriormente ficaria na tela da conta do usuário. A alteração é demonstrada na Figura 26. Um outro ponto importante que foi definido é inclusão do documento do carro para o Administrador poder recuperar esse documento quando for necessário, haja visto que atualmente, o departamento DERAC solicita uma cópia do mesmo.

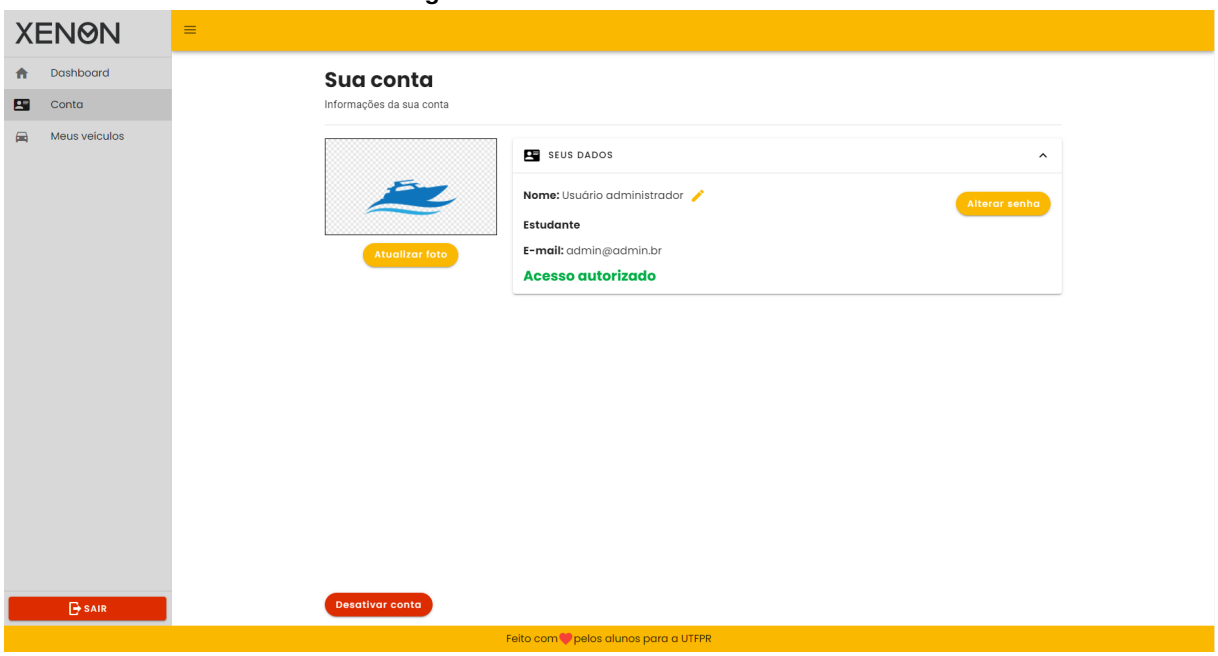
Por fim, a tela da conta do usuário abriga apenas as funcionalidades para exibir e alterar os dados da conta do usuário. É importante mencionar que o usuário pode alterar o nome, a senha e incluir um avatar na conta ou desativar sua conta. A Figura 27 demonstra a tela final da conta do usuário. Enfim, nesta *Sprints*, todas as histórias foram concluídas e houve a entrega do artefato conforme o planejado.

Figura 26 – Tela de listagem de carros cadastrados



Fonte: autor.

Figura 27 – Tela da conta de usuário



Fonte: autor.

6.2 SPRINT 2

Na Sprint 2 foi definida a implementação das histórias de ID 4[5], 5[6] e 6[7], porém, com alguns refinamentos. Rigorosamente, a história de ID 6 recebeu algumas alterações, o *Scrum Team* definiu que deveria ser criado mais duas funcionalidades, uma para bloquear o acesso do usuário na sua totalidade e outra para bloquear um carro cadastrado em sua conta. Outro ponto também definido na cerimônia foi implementar duas funcionalidades para manipular a conta do usuário e outra para manipular os carros do usuário. Logo as tarefas proposta para a história 6 foram a seguinte:

- **Tarefas**:

    1. Implementar a funcionalidade para o administrador poder bloquear o usuário completamente.
    2. Implementar a funcionalidade para o administrador bloquear o acesso a um carro do usuário.
    3. Implementar a funcionalidade para listar os usuários cadastrados.

Inicialmente, durante a análise do sistema, não foi definida a quantidade máxima de carros possíveis de serem cadastrados por um usuário. Porém, na cerimônia de refinamento, o *Product Owner* definiu que um usuário poderia ter mais de que um carro, sendo que a quantidade máxima foi definida para 5 (cinco) carros. Por conta disto, também foi revista a modelagem do banco de dados, como pode ser observado na Figura 28. Além de incluir estas funcionalidades para a liberação do acesso de veículos, o *Product Owner* definiu que deveria ser incluído mais uma funcionalidade da qual o Administrador poderia ou não aceitar o cadastro, esta nova funcionalidade foi marcada para ser refinada na *Sprint* posterior.

Como pode ser observado na Figura 28, principalmente nas tabela users e cars, foram incluídos um campos com o nome **"authorized_access"** em ambas as tabelas, da qual controla se o carro tem ou não acesso ao estacionamento. Por meio deste dado se pode então bloquear o usuário em sua totalidade ou apenas um carro do usuário cadastrado.

Na sequência, a história de ID 4 foi implementada, sendo que parte desta história foi realizada na *Sprint* anterior, como parte da história de ID 2, ou seja, com a implementação da tela com o formulário para incluir um operador ou editá-lo. Assim, na Sprint corrente, estas ações para editar e remover foram concluídas na tabela da lista de usuários. Após a implementação da funcionalidade para listar os usuários, também foram incluídas as ações para bloquear e des-

[5] Como administrador quero adicionar um novo motorista ao sistema, como atualizar seus dados ou removê-lo do cadastro.

[6] Como administrador quero adicionar uma única placa de carro a um motorista já cadastrado.

[7] Como administrador quero bloquear o acesso de um motorista já adastrado, temporariamente, assim como desbloqueá-lo, assim como ver a lista dos bloqueados.

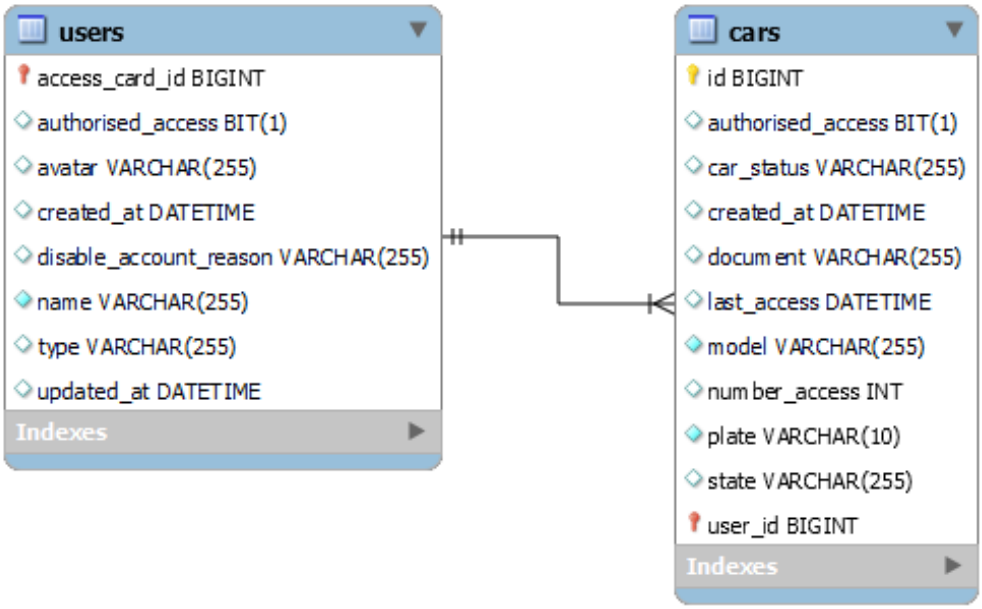
bloquear, bem como a informação de estado da conta do usuário para facilitar esta manipulação ao Administrador, tal como mostrado na Figura 29.

A partir da entrega desta implementação, o *Product Owner* solicitou uma tela de exibição para apresentar a lista de carros de cada usuário, o mesmo criou a história com prioridade alta, de valor 1, a história criada recebeu a seguinte descrição: **COMO** Administrador **QUERO** poder observar todos os carros cadastrados. Esta história recebeu o ID 12 e foi inserida no *Product Backlog*.

No decorrer da *Sprint*, a história de ID 5 foi analisada e foi removida da *Backlog*. Esta funcionalidade foi definida como inviável, dado que apenas o usuário que cadastrou o carro deveria então adicionar os carros da qual gostaria de incluir para autorizar.

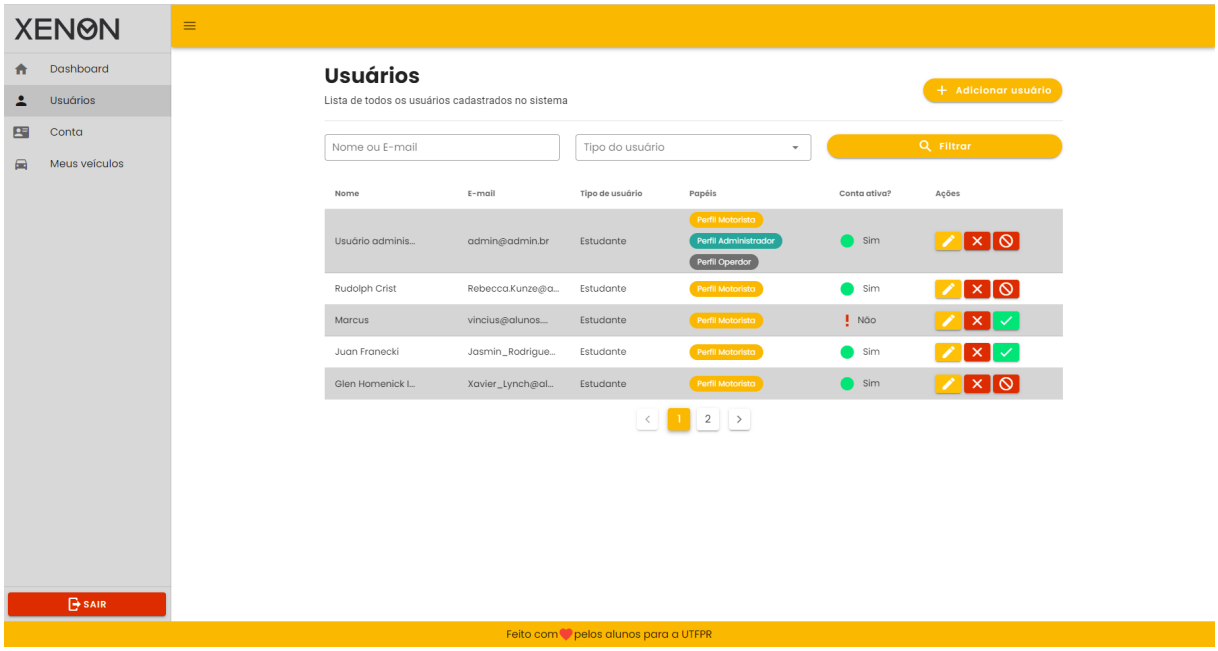
Durante a entrega da corrente Sprint, uma nova história foi criada pelo *Product Owner* referente à funcionalidade para listar os reconhecimentos recebidos na aplicação web, separado por usuários para questões de auditoria. Esta história recebeu prioridade alta e incluída com o ID 13. O *Product Owner* definiu a história com a seguinte descrição: **COMO** Administrador **QUERO** poder observar todos os reconhecimentos/acessos que o sistema obteve separado por usuários.

Figura 28 – Relacionamento Usuário com Carro



Fonte: autor.

Figura 29 – Lista de usuários cadastrados



Fonte: autor.

6.3 SPRINT 3

Nesta *Sprint*, as histórias escolhidas para serem implementadas foram as seguintes: 8[8], 12[9] e 13[10]; além de alguns refinamentos de histórias da *Sprint* anterior. Particularmente, uma questão envolvia a inclusão da história de ID 7, mas esta foi substituída pela história de ID 13, dado que a funcionalidade de ambas as histórias eram semelhantes em alguns pontos específicos. Entretanto, a história de ID 7 ficou obsoleta sendo necessário a sua substituição. Ao final da análise, o *Scrum Team* escolheu a história de ID 8 para iniciar a *Sprint*, dado que era necessário a estrutura de reconhecimentos para então listar os acessos com base nos reconhecimentos. A história então foi diluída nas seguintes tarefas:

- **Tarefas**:

  1. Modelar o banco de dados para incluir a tabela de reconhecimentos;
  2. Implementar na aplicação de reconhecimento o envio das placas reconhecidas;
  3. Implementar na aplicação de reconhecimento, o recebimento da requisição para enviar o sinal para GPIO da cancela;

[8] Como operador quero liberar o acesso a um motorista mediante a sinalização do sistema que ele tem autorização.

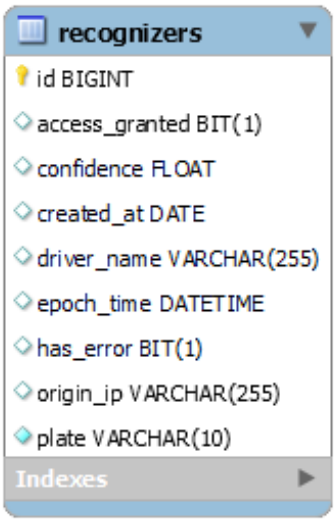
[9] Como Administrador quero poder observar quantidade de carros cadastros do usuário

[10] Como administrador quero poder verificar os acessos do do usuário e seus carros cadastrados no estacionamento.

4. Implementar na aplicação web o fluxo para identificar o usuário com base no reconhecimento e o envio para aplicação de reconhecimento a liberação de acesso.

A primeira coisa a se fazer foi a modelagem do banco de dados para salvar os dados do reconhecimento. Esta tabela recebeu o nome **recognizers** como mostrado na Figura 30. Após esta tarefa, foi implementado o fluxo para receber os resultados de reconhecimentos. Em seguida foi observado que era necessário guardar informações de erros relacionados ao reconhecimento, dado que a aplicação de reconhecimento não deveria conhecer o processo de validação do reconhecimento da *API*. Com base nesta regra, foi necessário um mecanismo de auditoria para saber o resultado do reconhecimento. Diante disto, foi pensado em um processo que seja executado quando uma exceção for lançada, que impeça concluir o processo de identificação do usuário com base no reconhecimento. Por exemplo, a aplicação de reconhecimento envia campos inválidos e a aplicação web não conseguia processar o reconhecimento recebido. Então, foi exposto este fato para o *Product Owner* que então solicitou uma nova história, além de outros pontos que necessitam de novas observações, da qual foi discutido a origem dos dados de reconhecimento. Até então, era esperado haver apenas uma fonte de dados, ou seja, apenas uma portaria. Este ponto fez com que o *Scrum Team* desenhasse uma solução que tenha uma ou mais fontes de reconhecimento, até mesmo para atender situações de uma cancela para entrada e outra para saída de veículos.

Figura 30 – Tabela de Reconhecimentos



Fonte: autor.

Por fim, foram criadas duas novas histórias de prioridade alta com os ID 14 e 15 e as seguintes descrições: **COMO** administrador **QUERO** cadastrar mais de uma portaria para ser gerenciada pelo Sistema Xenon e **COMO** administrador **QUERO** salvar dados de auditoria para quando ocorrer erros em uma ação de reconhecimento. Após a criação da história, foi discutido com o *Product Owner* substituir as histórias escolhidas para *Sprint* atual, no caso de ID 12 e 13,

para as recém-criadas, uma vez que as previamente escolhidas são impeditivas até a conclusão das histórias novas. O *Framework Scrum* permite que uma história seja bloqueada e ao mesmo tempo ligada com outra história, sendo ela dependente da uma outra a ser concluída. Deste modo, as histórias de ID 12 e 13 foram bloqueadas.

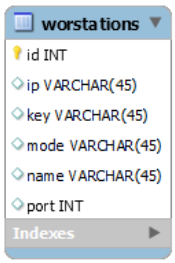
Após o aceite do *Product Owner*, as histórias de ID 14 e 15 foram incluídas na *Sprint* atual, sendo a de ID 14 a primeira a ser feita. Estas foram diluídas nas seguintes tarefas:

- **Tarefas**:

    1. Modelar o banco de dados para incluir uma tabela *workstations*;
    2. Implementar a funcionalidade para criar uma estação de trabalho;
    3. Implementar mecanismos para vincular o agente inteligente a estação de trabalho;
    4. Implementar mecanismos de segurança nas requisições para envio de informações de reconhecimento;
    5. Implementar a funcionalidade para remover uma estação de trabalho;
    6. Implementar a funcionalidade para editar uma estação de trabalho.

Para dar conta destes refinamentos, foi criada uma nova tabela referente às estações de trabalho, denominada **worstations**, como mostrado na Figura 31. O atributo **key** da entidade será então acrescida ao objeto inteligente para ele ser vinculado à Estação de trabalho criada, e todos os reconhecimentos enviados serão vinculados a mesma. Um filtro de requisição foi introduzido para avaliar a chave encaminhada na requisição.

Figura 31 – Tabela de Estação de trabalho

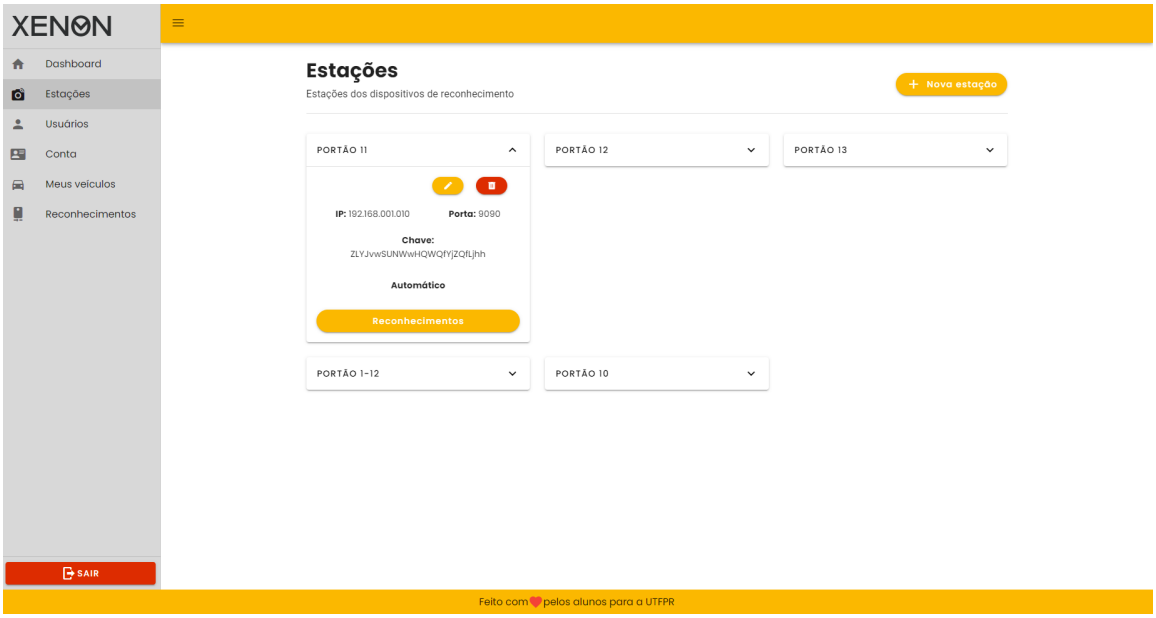


Fonte: autor.

Uma tela para listar as Estações de trabalho e um modal foram desenvolvidas para criar uma estação, como mostram a Figura 32 e Figura 33, respectivamente. Este modal apresentado é acionado na página de listagem de estações de trabalho.

Em relação à quantidade de estações de trabalho, ela não ultrapassará a quantidade de 50 unidades. Por conta disto, não foi necessária uma lista, sendo que o *Scrum Team* decidiu por criar uma exibição mais limpa. O formulário que foi utilizado para criar uma Estação de trabalho também foi utilizado para editar a mesma. Por último, foi concluída a tarefa para remover uma estação de trabalho.

Figura 32 – Tela para listar Estação de trabalho



Fonte: autor.

Figura 33 – Modal para criar Estação de trabalho

Fonte: autor.

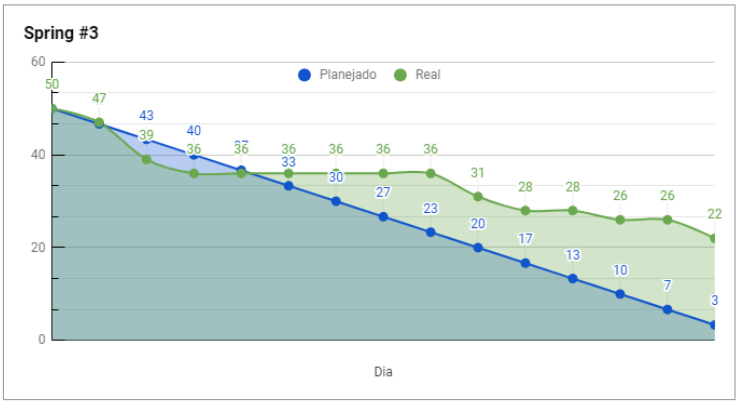
Após a conclusão da história de ID 14 foi iniciada a história de ID 15. Entretanto, esta história não foi concluída na *Sprint* atual, sendo ela transbordada para próxima *Sprint*. O desenvolvimento da história será dissertada na seção da próxima *Sprint*.

Ao final desta *Sprint*, as metas não foram alcançadas, principalmente por conta das alterações e refinamentos que impactaram o desenvolvimento da *Sprint*. Inicialmente, a *Sprint*

foi definida em 50 pontos com base na quantidade de tarefas a serem executadas. Todavia, considerando a teoria do o gráfico *Burn Down*, vale observar que cada *Sprint* tem um intervalo de 30 dias, sendo que no último dia, o ideal é que a pontuação termine em zero pontos a serem entregues. Esta linha ideal é apresentada na Figura 1 que demonstra o gráfico *Burn Down* e a tabela de dados de evolução de pontuação. Neste gráfico, também é apresentada a curva de desenvolvimento, pela qual se pode constatar o atraso. Nesta *Sprint*, houveram alguns dias sem desenvolvimento, entre o dia 8 e dia 16, que foi o tempo ocupado para pensar na solução do problema enfrentado, impactando diretamente na evolução da *Sprint*. Em seguida é possível observar a retomada do desenvolvimento, entre os dias 17 e 30, entretanto não foi suficiente, pois como pode ser observado na coluna Planejado da tabela na imagem Figura 1, que indica a quantidade de pontos a serem entregues no dia especificado, usando como referencia a coluna Dia, como por exemplo, no dia 20 para poder bater a meta da *Sprint* era necessário pontuar 20 pontos, porém a pontuação real entregue, como pode ser observado na coluna Entregas, foi de 5 pontos.

Gráfico 1 – Gráfico Burndown da Sprint 3

Burn down chart						
Real			Balanço			
Planejado	Entregas	Saldo	Dia	Planejado	Real	Faltam
3	3	0	1	50	50	47
4	8	-4	4	47	47	39
3	3	0	6	43	39	36
4	0	4	8	40	36	36
7	0	7	10	37	36	36
10	0	10	12	33	36	36
14	0	14	14	30	36	36
17	0	17	16	27	36	36
20	5	15	18	23	36	31
19	3	16	20	20	31	28
19	0	19	22	17	28	28
22	2	20	24	13	28	26
24	0	24	26	10	26	26
27	4	23	28	7	26	22
26	0	26	30	3	22	22



Fonte: autor.

6.4 SPRINT 4

A *Sprint* 4 foi iniciada com uma história transbordada da *Sprint* anterior, no caso a história de ID 15[11]. Além desta história, ainda foram incluídas as histórias de ID 12[12] e 13[13] que foram bloqueadas na *Sprint* anterior. A história de ID 15 foi diluída nas seguintes tarefas:

- **Tarefas**:

[11] Como administrador quero quando um erro ocorre em um reconhecimento, para seja salvo para poder ser feita uma auditoria sobre o caso.

[12] Como Administrador quero poder observar quantidade de carros cadastros do usuário

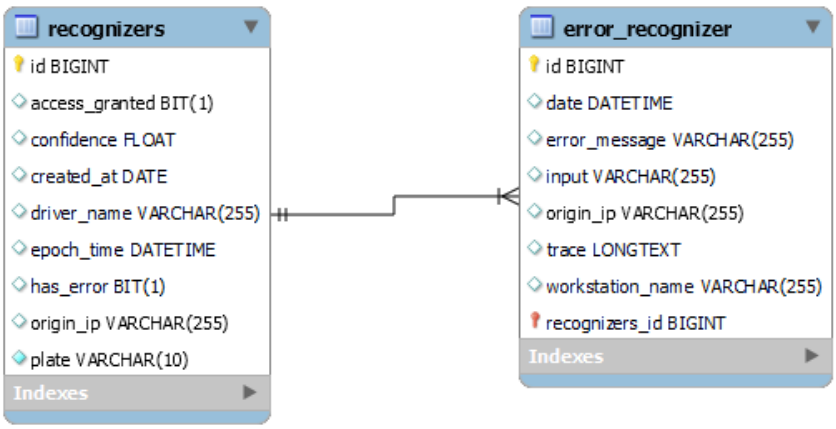
[13] Como administrador quero poder verificar os acessos do do usuário e seus carros cadastrados no estacionamento.

1. Modelar o banco de dados para uma tabela que salve os dados dos erros;
2. Implementar o fluxo para quando acontecer um erro, salvar na base de dados;
3. Alterar a tela que exibe os resultados dos reconhecimentos de placas.

A modelagem do banco de dados foi a primeira tarefa a ser executada, as alteração do banco de dados foram a inclusão de uma nova tabela e a realização do vínculo com a tabela de reconhecimento.

Vale salientar que a coluna na tabela de reconhecimento, no caso **only_error**, a qual foi introduzida na história de ID 8, será utilizada para sinalizar quando um reconhecimento não foi concluído por conta de um erro. Também é importante ressaltar que a coluna **input** da tabela **error_recognizer** é usada para salvar o *payload* (carga útil da requisição), tal como apresentado na Figura 34.

Figura 34 – Tabelas para reconhecimento e erro de reconhecimento

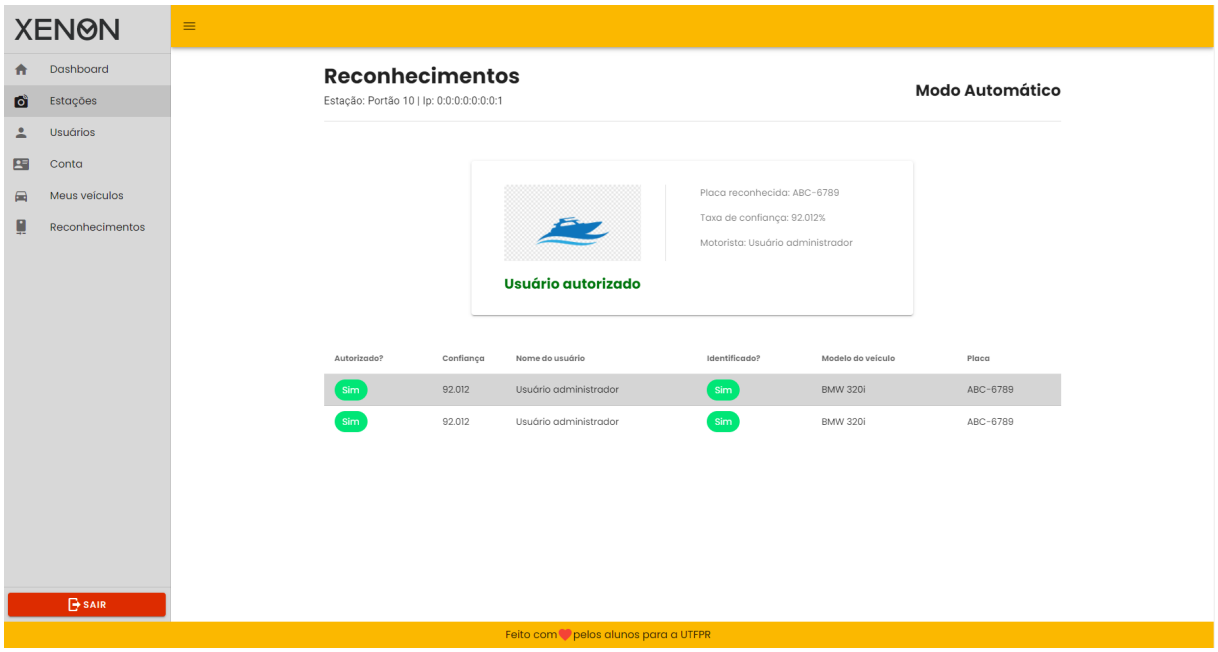


Fonte: autor.

O fluxo para capturar os erros do processo de reconhecimento foi implementado e adicionado ao fluxo de recebimento dos dados do reconhecimento de placas. Após isto, foi alterada a tela de reconhecimento que também foi criada na história de ID 8. Mais precisamente, a tela foi organizada em duas abas, ou seja, os reconhecimentos com erro e sem erro. Na aba de reconhecimento com erros, foi adicionado um botão que exibirá um modal que apresenta os detalhes do erro.

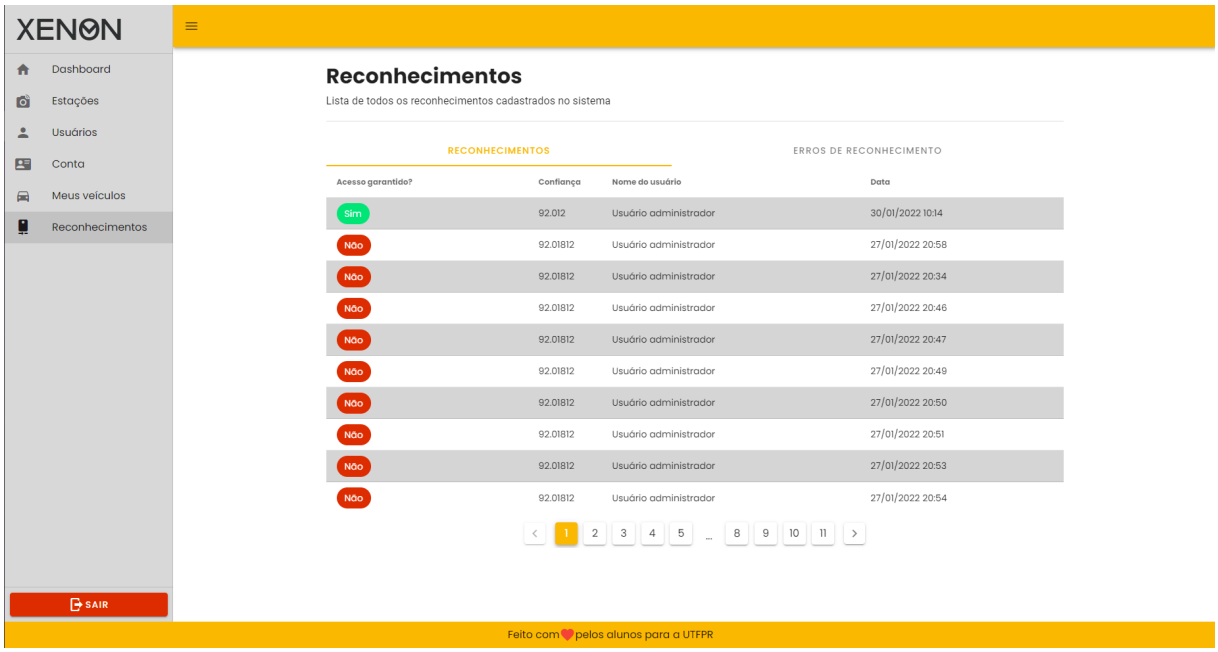
A conclusão desta história completa todo o fluxo do reconhecimento de placas veiculares. Nas Figuras 35, 36, 37 e 38 são demonstradas as telas construídas na aplicação de apresentação para estas funcionalidade.

Figura 35 – Tela de reconhecimento



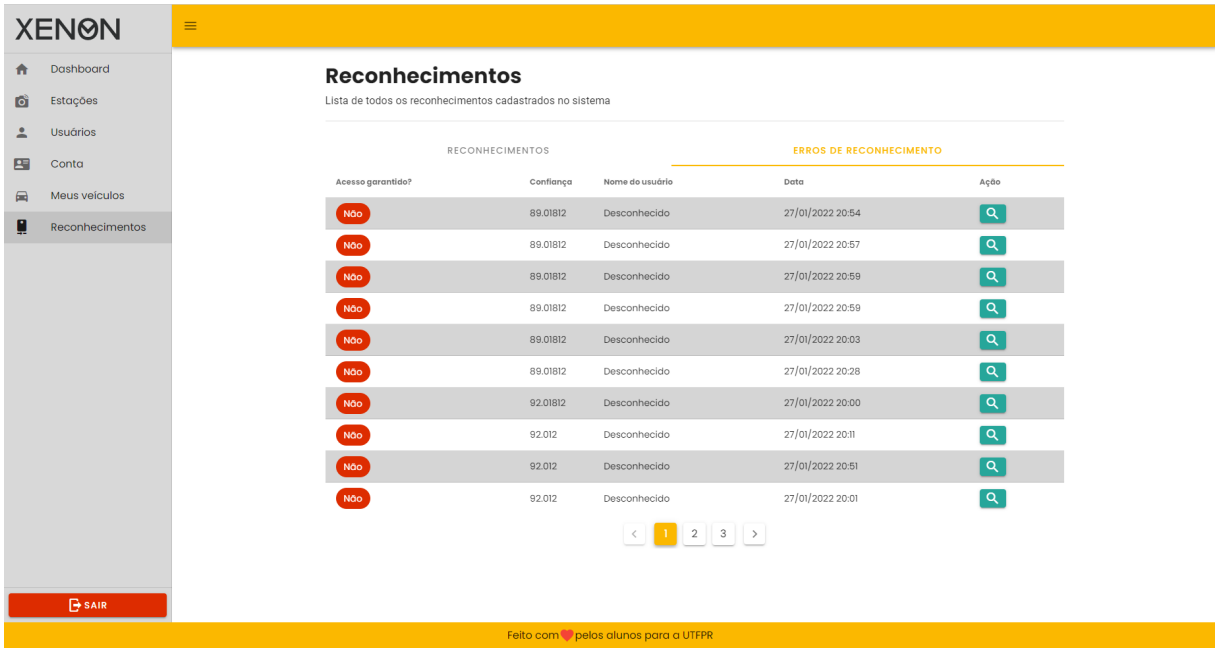
Fonte: autor.

Figura 36 – Aba de listagem de reconhecimentos sem erros



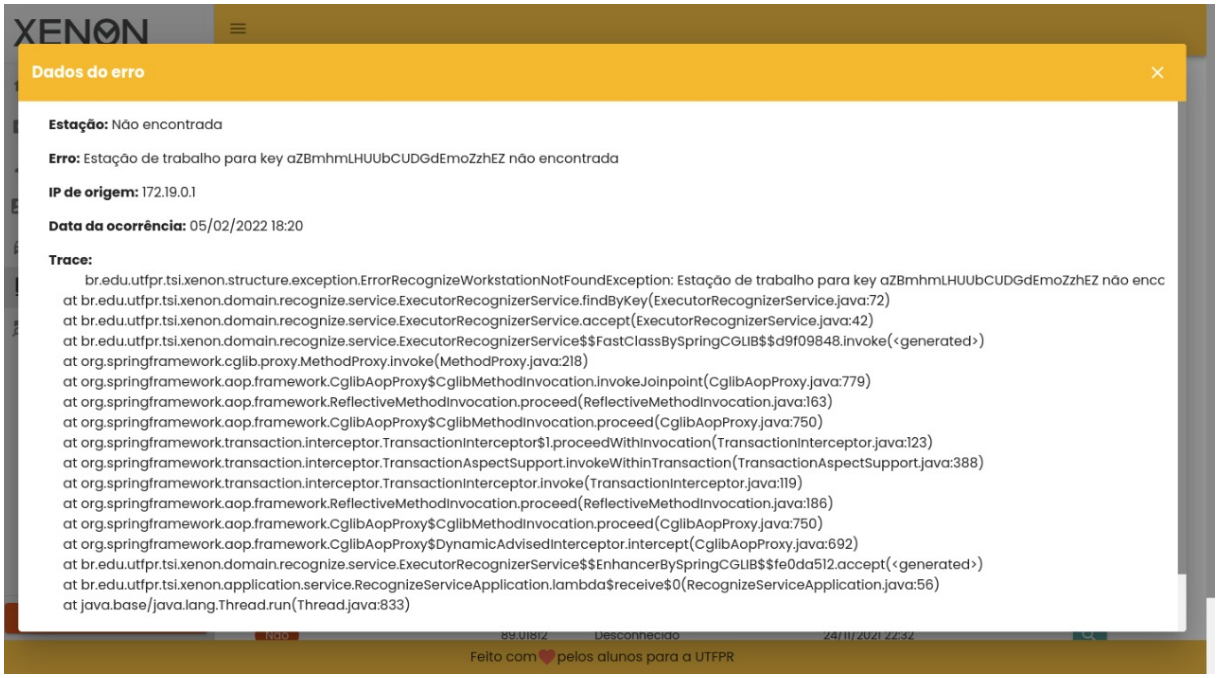
Fonte: autor.

Figura 37 – Aba de listagem de reconhecimentos com erros



Fonte: autor.

Figura 38 – Modal que apresenta o detalhe do erro



Fonte: autor.

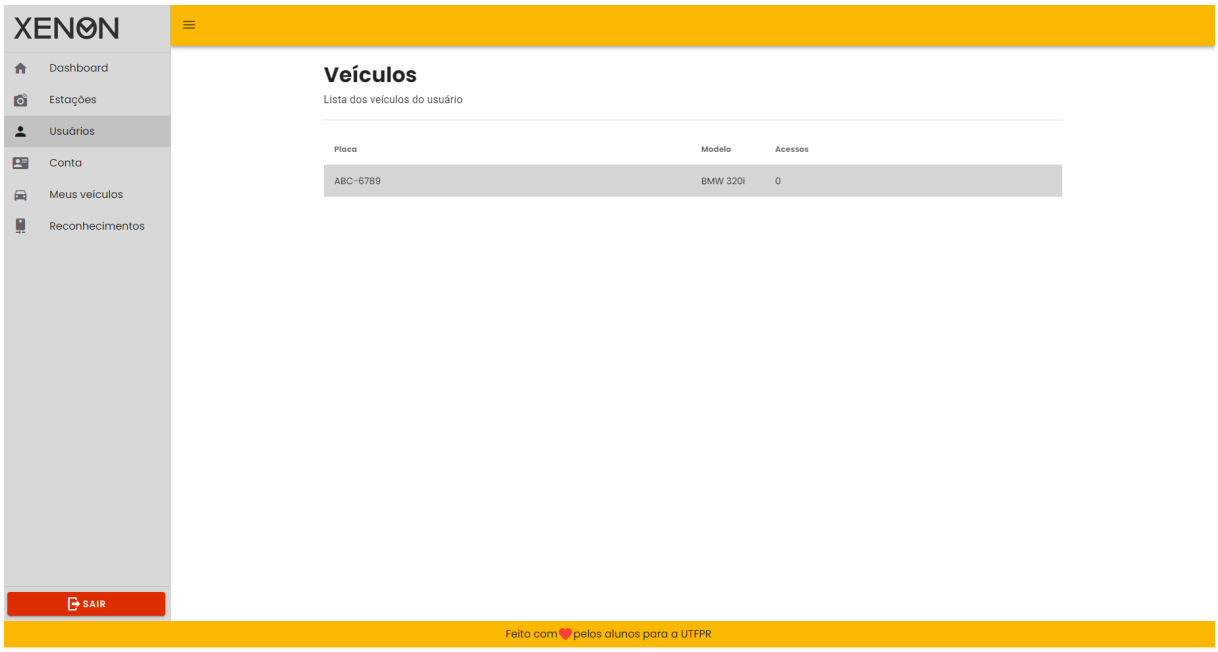
Em continuidade à *Sprint*, ocorreu a conclusão da história de ID 12. Esta foi discutida com o *Product Owner* para incluir um botão na tabela de usuários, que dá acesso a tela que exibe os carros cadastros do usuário, a fim de, facilitar o acesso ao administrador. Diante disto, a tabela foi alterada e uma tela foi construída para exibir a lista de carros. Após este refinamento, a história foi diluída nas seguintes tarefas:

- **Tarefas**:

    1. Implementar a funcionalidade para mostrar a lista de carros de um usuário;
    2. Alterar a tela de usuários, para incluir o botão para acessar a tela de lista de carros do usuário.

A conclusão desta história resultou na tela para exibir a lista de carros do usuário, como mostrado na Figura 39.

Figura 39 – Tela que exibe a lista de carros cadastrados do usuário



Fonte: autor.

Após concluir a história de ID 12, foi observado que era necessário um fluxo para avaliar o cadastro do carro. Isto é importante para o administrador recusar ou não o cadastro de um carro, tal como, em situações em que o documento enviado pelo usuário é invalido ou não coincide com a placa informada. Diante disto, a história de ID 16 foi proposta com prioridade baixa com a seguinte descrição: **COMO** administrador **QUERO** poder avaliar e recusar ou aprovar o cadastro do carro no sistema. Esta história foi adicionada ao *Backlog* para posteriormente ser refinada e implementada.

Por fim, a história de ID 13 foi concluída e então diluída nas seguintes tarefas:

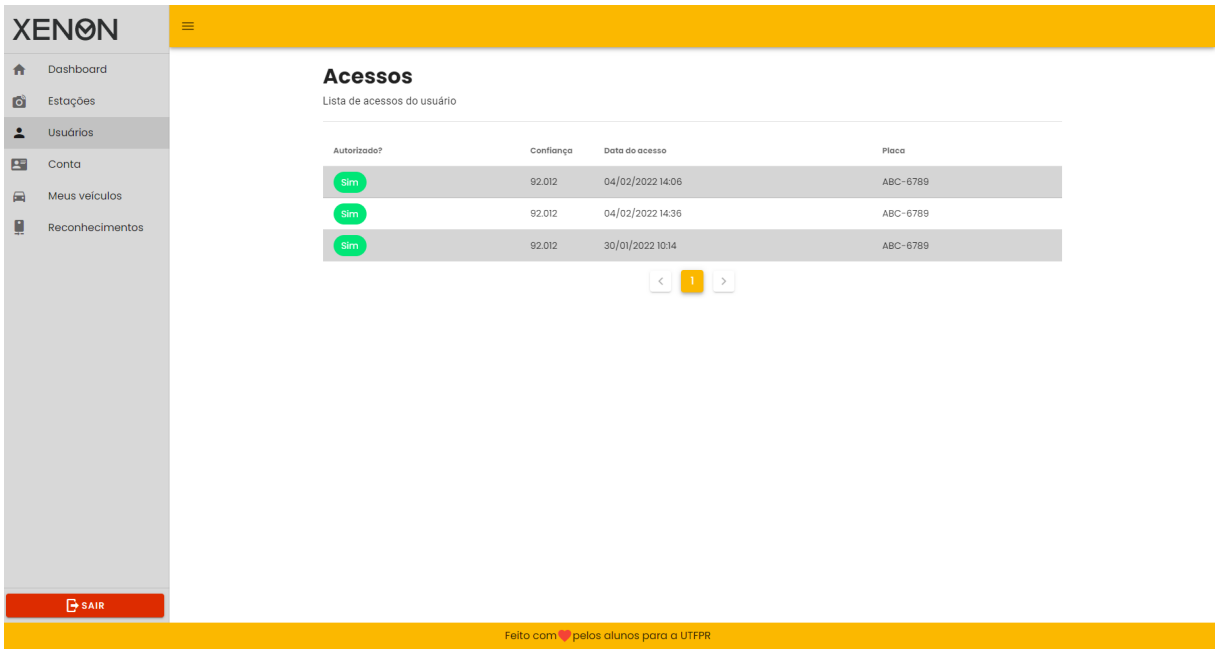
- **Tarefas**:

    1. Implementar a funcionalidade para mostrar a lista de acessos do usuário;
    2. Alterar a tela de usuários para incluir um botão para dar acesso a tela de listagem de acessos do usuário.

Esta história seguiu o mesmo padrão da história anterior, ou seja, implementar a funcionalidade sem alterações em modelagem de banco de dados ou outras funcionalidades já existentes. Também foi adicionado um botão para acionar a ação de exibir os acessos/reconhecimentos no sistema Xenon. A tela construída pode ser observada na Figura 40.

Com as histórias concluídas, a *Sprint* foi finalizada sem problemas, diferentemente da anterior, quando houveram alguns percalços que impediram a *Sprint* ser concluída como o planejado.

Figura 40 – Tela que exibe a lista de acessos/reconhecimentos do usuário



Fonte: autor.

6.5 SPRINT 5

A *Sprint* 5 foi composta pela a implementação da história de ID 16[14] e outras que serão apresentadas no decorrer desta seção. Mais precisamente, a história de ID 16 foi refinada e desmembrada nas seguintes tarefas:

- **Tarefas**:

[14] Como administrador quero poder avaliar e poder recusar ou o carro cadastrado no sistema pelo motorista.

1. Implementar a máquina de estado para cadastros de carros;
2. Implementar a funcionalidade para aprovar/reprovar um cadastro de carro.

Após a conclusão da história 16, foi identificado que era necessário um mecanismo de notificação para o usuário administrador quando novos cadastros forem incluídos na lista para avaliação. Até o momento, o usuário administrador teria que periodicamente acessar a página de carros no estado de "aguardando avaliação"para observar se houveram novos cadastros. Diante disto, foi exposto para o *Product Owner* que então criou uma nova história com baixa prioridade com a seguinte descrição: **COMO** administrador **QUERO** ser notificado quando um cadastro de carro for incluído no sistema para ser avaliado.

A nova implementação gerada pela história é mostrada na Figura 41. Nesta tela é permitido avaliar o cadastro do novo carro, para então ser liberado o acesso ao estacionamento.

Figura 41 – Tela para avaliar o cadastro do carro

Fonte: autor.

Ademais, o *Scrum Team* observou que não havia sido criada uma história para a funcionalidade de recuperar a senha da conta do usuário. Este caso foi exposto ao *Product Owner*, que então criou uma nova história com prioridade média de ID 18 com a seguinte descrição: **COMO** usuário do sistema **QUERO** recuperar a minha senha quando eu esquecer.

Também, o *Product Owner* solicitou uma funcionalidade para automaticamente desativar a conta do discente quando este completar a graduação. Esta funcionalidade foi então proposta para o time de desenvolvimento avaliar e propor uma solução para o caso. Não foi criada uma história de usuário para esta situação.

Ainda, o *Product Owner* solicitou algumas história de prioridade média, que são elas: a de ID 19 com a descrição: **COMO** administrador do sistema **QUERO** ter as informações de

quantidade de usuários cadastrados separados por tipos; a de ID 20 com a descrição: **COMO** administrador do sistema **QUERO** ter as informações de quantidade de acessos de uma semana; a de ID 21 com a descrição: **COMO** administrador do sistema **QUERO** ter acesso rápido às Estações de Trabalho e a quantidade de reconhecimentos que cada uma teve.

As histórias propostas neste *Sprint* pelo *Product Owner* de prioridade média foram refinadas e então combinadas com o *Scrum Team* para serem introduzidas nas Sprint posteriores, exceto a história de ID 18 que foi decidido ser adicionada na *Sprint* atual. Sendo assim, ela foi diluída nas seguintes tarefas:

- **Tarefas**:

    1. Implementar a funcionalidade para recuperar a senha;
    2. Implementar o fluxo para envio de nova senha para email.

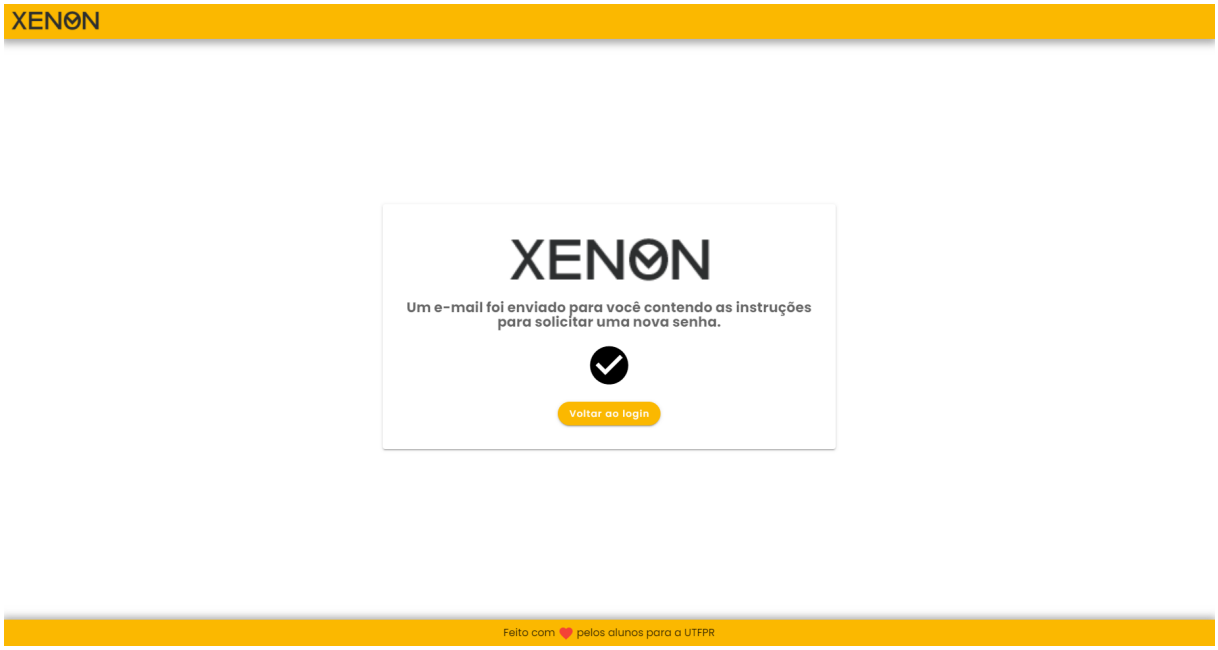
A implementação da funcionalidade de redefinição de senha, como foi chamada, gerou duas telas: uma para receber o email do usuário para então enviar as instruções para criar uma nova senha e outra para notificar o próximo passo, respectivamente as Figuras 42 e 43 demonstram o resultado.

Após preencher o email, caso válido, o sistema envia um link para notificar o usuário da tentativa de redefinir a senha, assim como mostrado na Figura 44. Este link apresenta uma vida útil de 5 horas. Este link então é direcionado para uma página do sistema para então iniciar o processo de redefinição de senha, tal como consta na Figura 45. Enfim, o *Scrum Team* decidiu que o sistema deve criar uma senha aleatória que será enviada por email ao usuário, finalizando o fluxo de redefinição de senha, conforme a Figura 46.

Figura 42 – Tela de formulário para email

Fonte: autor.

Figura 43 – Tela das informações do próximo passo

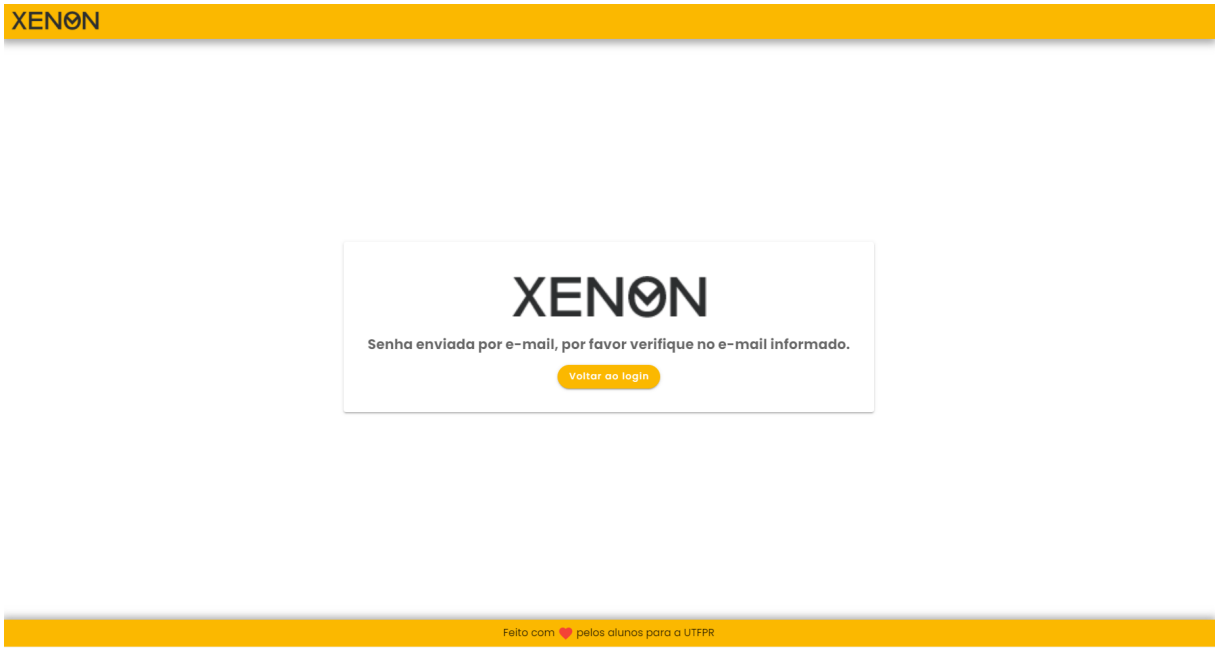


Fonte: autor.

Figura 44 – Corpo do email para solicitar uma nova senha

Fonte: autor.

Figura 45 – Tela para solicitar uma nova senha



Fonte: autor.

Figura 46 – Corpo do email contendo a nova senha

Fonte: autor.

6.6 SPRINT 6

A *Sprint* foi formada pela implementação das histórias de ID 19[15], 20[16] e 21[17]. A primeira a ser desenvolvida foi a de ID 19. Em princípio, foi pensando em criar uma única tela para exibir as informações da história, sendo que esta abordagem foi exposta para o *Product Owner* que sugeriu a ideia de construir uma tela de dashboard que será exibida no acesso do usuário Administrador ao sistema. Com este entendimento entre o *Scrum Team* e o *Product Owner*, e para facilitar o desenvolvimento da *Sprint*, todas as histórias foram diluídas de uma única vez, da qual resultou nas seguintes tarefas:

- **Tarefas**:

    1. Implementar uma tela de dashboard para o administrador;
    2. Implementar a funcionalidade de exibir a quantidade de usuários cadastrados;
    3. Implementar a funcionalidade para exibir reconhecimentos por semana;
    4. Implementar a funcionalidade para exibir a quantidade de carros separados por estado do cadastro, que são eles: ativo; negado; bloqueado ou em análise.

Vale mencionar que no decorrer do desenvolvimento da *Sprint*, foi notada uma alta carga de processamento no banco de dados, sendo que as consultas das referidas histórias tem uma alta carga de recurso computacional. A partir desta constatação, o *Scrum Team* observou que essas informações não precisavam ser atualizadas constantemente. Sendo assim, foi proposto um sistema de cache, sendo portanto utilizado o Redis.

Após esta implementação, o *Scrum Team* fez uma avaliação das requisições de recursos que poderiam ser também armazenados em cache. É importante mencionar que não foi necessário criar uma história para esta tarefa, pois foi definida como melhoria de sistema e feita na *Sprint* atual.

Ao final da avaliação, foi então decidido que os recursos da lista de usuários, estações de trabalho e lista de carros seriam colocados em cache. Após alguns testes de avaliação, o *Scrum Team* chegou em um valor de vida útil do cache de 5 minutos. Os testes levaram em consideração a diminuição e aumento de recurso computacional, memória e CPU, do servidor de banco de dados. Os testes ocorreram ao longo de uma hora, com leituras e escritas periódicas e aleatórias com 5 usuários simultaneamente. A conclusão foi que acima de 5 minutos de vida útil do cache, os resultados eram muito menores, dado que, a cada escrita o cache era deletado na sua totalidade ou parte dele, levando ao aumento de processamento para este caso.

Ao final da *Sprint* foi obtida uma tela de dashboard, conforme a Figura 47, como resultado das histórias mencionadas.

[15] Como administrador quero ter as informações de quantidade de usuários cadastrados separados por tipos.
[16] Como administrador quero ter as informações de quantidade de acessos por semana.
[17] Como administrador quero ter acesso rápido às Estações de Trabalho e a quantidade de reconhecimentos que cada uma teve.

Figura 47 – Tela de dashboard

Fonte: autor.

6.7 SPRINT 7

As história escolhida para ser implementada foi a de ID 17[18] que foi desmembrada nas seguintes tarefas:

- **Tarefas**:

    1. Implementar as notificações para o usuário que tem novos carros aguardando a análise, para ser ou não aceito o cadastro;
    2. Implementar na tela um ponto de atenção quando houver uma notificação.

Na análise para implementar a funcionalidade foi observado que deveria haver um intervalo entre uma notificação e outra. Neste caso, era necessário uma estrutura de *Push Notification*, mas, a implementação estava complexa para ser usada apenas em uma única funcionalidade. Diante deste problema, foi então definido que a solução menos complexa utilizando websocket, tecnologia já utilizada na aplicação, e agendador de tarefas, comumente chamado de *Job* (Trabalho em inglês). O Framework Spring já tem mecanismo de websocket pronto, apenas era necessário ativar e liberar para compilação. A solução então foi criar um *Job* que em um intervalo de cinco minutos faz uma consulta para verificar se existe ou não algum cadastro de carro aguardando avaliação do administrador. A consulta é barata em termos de processamento para um servidor de banco de dados, uma vez que no primeiro dado encontrado, o fluxo é interrompido, sem a necessidade de verificar todos os dados.

[18] Como Administrador quero ser notificado quando um cadastro de carro foi incluído no sistema para ser avaliado.

Para a tela, foi adicionado um canal de Websocket que quando receber uma mensagem, renderiza um ícone de sino. Quando o usuário clica neste ícone, ele será redirecionado para tela de avaliação de cadastro de carro.

Ao final da *Sprint*, o *Product Owner* solicitou duas funcionalidades no sistema, uma para desativar a conta do aluno que já concluiu o curso no campus de forma automática, ou seja, sem a necessidade de ação do administrador e outra para criar uma conta com tempo de vida útil, por exemplo, com a duração de um dia. Posto isto, o *Scrum Team* expôs algumas possíveis soluções, que em conjunto com o *Product Owner*, as seguintes histórias foram criadas: as histórias de ID 22 e 23 com prioridade baixa. Estas histórias apresentam as seguintes descrições, respectivamente: **COMO** administrador do sistema **QUERO** que os cadastros de alunos sejam desativados com o aluno após a conclusão do curso; **COMO** administrador do sistema **QUERO** quero cadastrar um usuário motorista e configurar um tempo para a conta ficar ativa. Estas histórias foram adicionadas no Backlog.

O tempo restante da *Sprint* foi utilizado para treinar o algoritmo ALPR da biblioteca OpenALpr.

6.8 PROCESSO DE CONFIGURAÇÃO E TREINAMENTO INCREMENTAL E TRANSVERSAL ÀS SPRINTS

O reconhecimento da placa, como explicada nas seções anteriores, foi executado dentro do objeto inteligente do Raspberry PI. Para esta tarefa, o algoritmo ALPR utilizado foi o da biblioteca OpenALPR. Este desenvolvimento foi dividido em duas fases:

- Instalação e configuração
- Aprendizagem;

Na fase de instalação, a biblioteca OpenALPR foi integrada a uma aplicação JavaScript que tem como responsabilidade atuar como um observador do OpenALPR, tal como como explicado na seção 2.2. Para obter um melhor desempenho e qualidade, algumas configurações e ajustes finos foram necessários com base nas informações da documentação.

O treinamento consistiu em ensinar o agente, ou seja, o algoritmo ALPR a reconhecer os padrões de placas de carros no modelo brasileiro tradicional e do Mercosul. Para poder segmentar os caracteres, o agente foi treinado com a técnica de aprendizagem de máquina supervisionada em duas etapas. Este treinamento consistiu de aproximadamente 200 a 300 imagens de placas de carros no modelo Mercosul, pois a biblioteca já tem suporte para o padrão antigo, com um tempo aproximado de 20 horas de processo de aprendizagem. Este processamento foi importante para reconhecer os padrões de localização da placa do carro no *Stream* de video. Assim, o algoritmo conseguiu separar a placa do carro através da segmentação por borda, se-

guindo para reconhecer os caracteres da placa segmentando-os um a um através da técnica de segmentação por corte.

Por fim, foram realizados os testes de comportamento e desempenho do agente para melhorar a segunda fase de treinamento, onde o agente foi submetido à avaliação de reconhecimento. Mas desta vez com *Stream* de vídeo para comprovar se o agente conseguia identificar com uma boa performance as placas de carro. Este processo para treinamento do agente foi iniciado antes das *Sprints* e foi até o final da última *Sprint* desenvolvida, haja vista que é um processo relativamente demorado. O fato de adquirir as imagens de placas para padrão antigo requer um tempo relativamente grande.

É importante mencionar que ocorreu a dificuldade de encontrar imagens de placas do padrão Mercosul para treinar o algoritmo, uma vez que a biblioteca OpenALPR, até a data do referido trabalho, tem suporte apenas para o padrão antigo, ou seja, no formato **AAA-1234**. Considerando esta complexidade no treinamento para placas nos modelos Mercosul, não foi possível ter um treinamento efetivo para o algoritmo, com milhares de imagens como sugere a documentação do OpenALPR. Portanto, este problema resultou em taxas de confiabilidade menores para o modelo em questão, embora essa taxa não tenha sido impactante para o bom funcionamento do OpenAPLR. Porém, nos testes, foi evidente a diferença de resultados, como será apresentado no capítulo **??**.

6.9 CONSIDERAÇÕES FINAIS

Nesta seção serão apresentados alguns fatores e considerações referentes ao desenvolvimento da solução. Basicamente, é apresentado como foram utilizadas as ferramentas de desenvolvimento, como por exemplo, o Git de acordo com um fluxo de trabalho definido pelo autor e as dificuldades enfrentadas no decorrer do desenvolvimento.

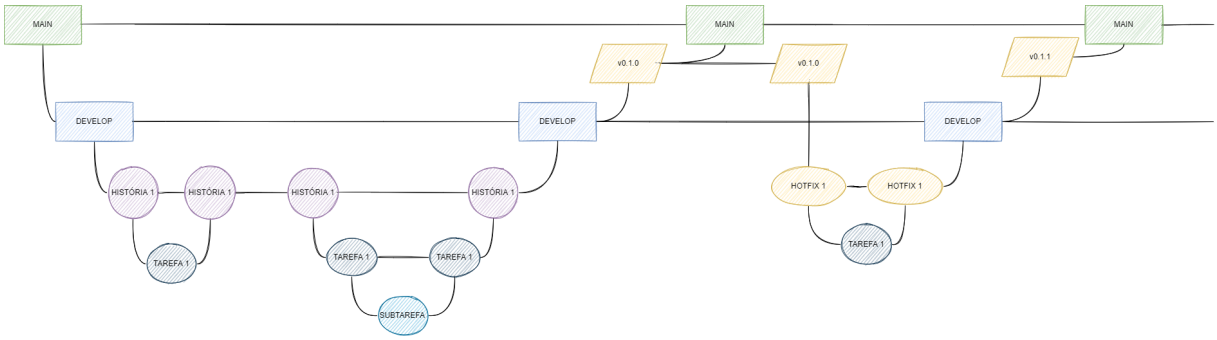
O código-fonte do projeto foi construído usando práticas de versionamento de código, ou seja, a ferramenta de versionamento de código Git. Mais precisamente, foi construído um repositório para abrigar o código-fonte na plataforma Github.

Neste repositório, foram criadas duas ramificações, comumente chamadas de *branch*, as quais foram chamadas de **main e develop**. O branch develop é o ponto de bifurcação para o versionamento do código-fonte na fase de desenvolvimento, ao passo que a *branch* main recebe o código fonte estável, isto é, a cada *Sprint* um artefato é empacotado, marcado com uma tag de versão que então é extraído da *branch* develop.

Para cada história, uma nova *branch* foi criada a partir de develop, esta *branch* é comumente chamada de *branch feature*. Ao final da implementação das funcionalidades de uma dada história, o código era mesclado ao *branch* develop. Em cada *branch* feature, também ocorreu de serem criadas outras ramificações para cada tarefa da respectiva história.

Ao final de uma Sprint, após a análise do código e validação dos testes, as histórias completas foram empacotadas em um tag de versão. Caso uma bug seja encontrado após a entrega do artefato, a ramificação para a correção do bug é denominada de *hotfix* e esta tem com base a tag de versão. Após a conclusão da correção, esta é então mesclada à *branch* develop, que então é empacotada e mesclada na *branch* main. O fluxo de trabalho é apresentado na Figura 48.

Figura 48 – Fluxo de versionamento de código



Fonte: autor.

Para manter a consistência e qualidade do código fonte, a ferramenta Travis-CI foi utilizada para fazer a análise da build e dos testes. Também foi utilizada a ferramenta Codecov para validar a cobertura dos testes. O fluxo deste trabalho foi configurado para ocorrer automaticamente a cada *Pull Request* realizado no GitHub, ocorrendo as validações pelo Travis-CI e Codecov e na sequência o processo de mesclagem para *branch* main. Os fluxos de entrega de cada artefato mencionado anteriormente foram executados em todas as *Sprints*.

É importante mencionar que inicialmente a aplicação foi desenvolvida como uma aplicação web tradicional, que gerava as páginas HTML no servidor e as devolvia para apresentação no lado cliente, ou seja, no navegador. Esta técnica é comumente chamada de *Server Side Render*. À medida que o desenvolvimento decorreu, surgiu a ideia de construir um aplicativo mobile para também ser cliente da API da aplicação. Foi então pensado em separar a aplicação web em Frontend e Backend para no futuro utilizar a tecnologia *PWA* para poder compilar para um aplicativo mobile sem muito esforço. Diante disto a aplicação web foi refeita para separar em Frontend e Backend.

Outro ponto importante mencionar é que houve uma grande pausa no desenvolvimento do referido trabalho, da qual foi iniciado em 2019. Por diversos motivos, inclusive e principalmente no que se refere à pandemia global vivenciada, esta pausa levou a uma grande dificuldade na retomada dos trabalhos.

Por fim, no que se refere à aplicação web, afim de manter a qualidade do código e cobertura de testes, foram utilizadas as ferramentas de CI-CD (Continuos Integration e Continuos Delivery, ou seja, Integração Contínua e Entrega Contínua) basicamente é um conjunto de práticas que combinadas automatizam a entrega dos artefatos de um projeto, para manter a qualidade da aplicação. Para demonstrar publicamente a qualidade da aplicação no repositório

GitHub, foram utilizadas as melhores técnicas possíveis de mercado e da academia, tal como a abordagem *evergreen*, que nada mais é do que *badge*(etiqueta) que comprova a qualidade do código da aplicação, como mostrado na Figura 49. Propriamente, o *badge build* refere-se a compilação da aplicação e se todos os testes passaram, ao passo que *maintainbility* define a qualidade do código, por exemplo se tem código duplicado, *code smell* (código mal cheiroso em uma tradução livre, quer dizer código mal escrito), e por fim, o *codecov* demonstra a cobertura de teste da aplicação.

Figura 49 – Evergreen

Fonte: autor.

7 RESULTADOS

ara comprovar a aplicabilidade das aplicações de reconhecimento e da aplicação web, foram feitos alguns testes em ambientes controlados. Mais precisamente, os testes foram feitos em ambiente interno simulado utilizando-se de imagens de placas apresentadas em um monitor e outros em ambiente externo, próximo do real, com automóveis dispostos diante do sistema desenvolvido. Os testes externos avaliam o fluxo de reconhecimento de ponta a ponta, do momento em que o carro pará até a notificação em tela.

Este capítulo está dividido da seguinte maneira: na seção 7.1 são apresentados os testes internos e na seção 7.2 são apresentados os testes externos. Por fim, na seção 7.3 são apresentadas as considerações finais.

7.1 TESTES INTERNOS

Os testes internos ocorreram utilizando 312 imagens de carros, sendo essas divididas nos modelos de placas brasileiras no padrão antigo (composta por 3 letras seguidas por 4 números) e Mercosul (composta por três letras seguidas, um número, uma letra e depois mais dois números). As imagens foram adquiridas de forma aleatória na Internet. É importante mencionar que as placas adquiridas não foram relacionadas a identidade dos seus donos, por questão de privacidade, elas não serão mostradas neste referido trabalho. Também é importante mencionar que mesmo a aplicação enviando requisições para a aplicação web, das 312 imagens que correspondem as placas, apenas 20 foram utilizadas para criar contas de usuários e carro na aplicação web.

Nos testes internos, o Raspberry foi fixado em uma altura de 40 cm a partir do solo. Para exibir as imagens, foi utilizado uma TV *Wide* de 29 polegadas, que foi posto à direita do Raspberry em um ângulo de aproximadamente 25 graus, como demonstrado na Figura 1. As posições de ambos os objetos foram deliberadas para simular a instalação do objeto inteligente fixado ao lado de uma cancela. As imagens foram exibidas de forma aleatórias em intervalos de 30 segundos aproximadamente entre uma e outra. A cada reconhecimento foi registrado a confiabilidade e o tempo do processamento. O tempo total do experimento foi de aproximadamente três horas.

Também foram feitos testes de comunicação para validar que a aplicação web e a aplicação de reconhecimento tinham uma comunicação fluída. Para estes testes, foi adicionado um relé GPIO 12 na porta lógica utilizada para este fim, da qual a aplicação de reconhecimento está mapeada para enviar um sinal. Para validar esta comunicação, foi feito um reconhecimento com a estação de trabalho configurada no modo manual e posteriormente no modo automático. No modo manual, a cancela é ativada manualmente por um operador humano, já no modo automático, a cancela é ativada automaticamente após o reconhecimento autenticado de uma placa.

Fotografia 1 – Posicionamento do Raspberry

Fonte: autor.

No modo automático, quando o reconhecimento for autenticado corretamente, será enviado uma requisição para a aplicação de reconhecimento mandar um sinal para GPIO. Para satisfazer esta ação, foi instalado um relé para ser acionado quando a requisição for bem sucedida. Na Figura 2 é demonstrado o relé sendo ativado, além de **logs** para escrever no console o momento do recebimento da requisição. Já no modo manual, para ativar o relé, é seguindo o processo anteriormente mencionado, mas um botão será renderizado na tela da aplicação web para iniciar a ação. As Figuras 50, 51 e 52 demonstram o resultado destes testes. Mais precisamente, do lado direito da imagem está a tela de reconhecimento da aplicação web e do lado esquerdo os logs de console da aplicação de reconhecimento.

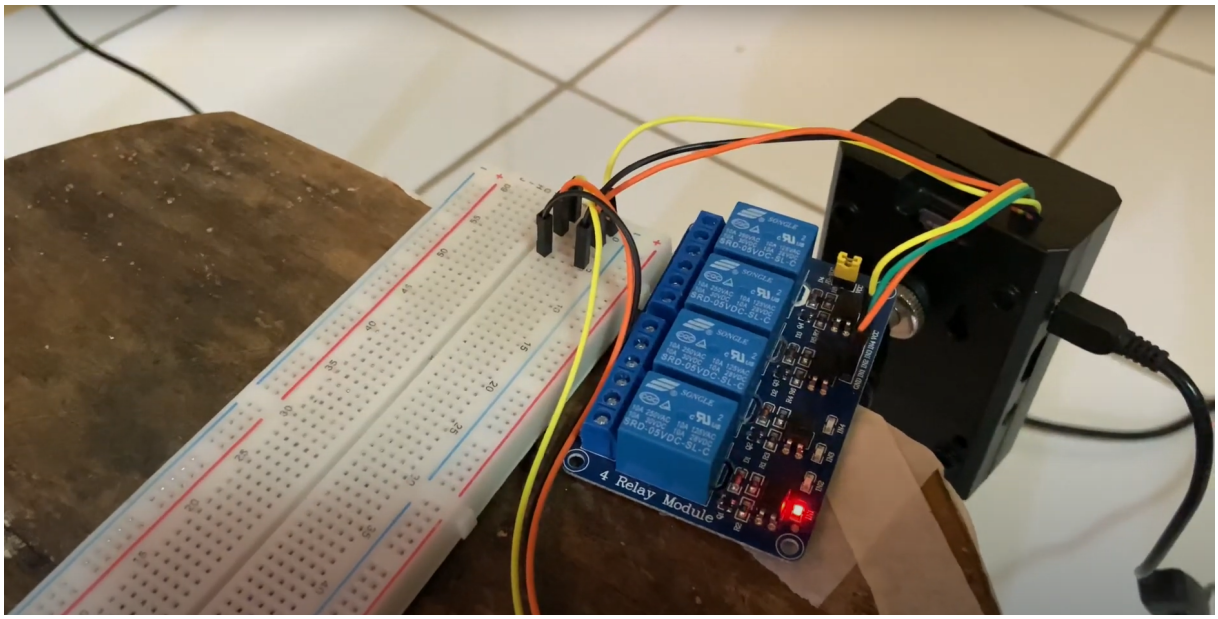
Os resultados obtidos, como mostrado na tabela 5, foram separados pelos padrões das placas Mercosul e padrão antigo. Vale salientar que todas as placas foram fielmente reconhecidas pelas aplicação de reconhecimento. O tempo em média para cada reconhecimento em relação ao padrão antigo em detrimento do padrão Mercosul, se dá ao fato do cálculo do padrão antigo ser o primeiro na sequência. Na biblioteca OpenALPR, após encontrar a localização da placa e extrair os caracteres, ela faz a validação da sequência de caracteres obtidos para validar se a sequência corresponde com o padrão antigo ou Mercosul.

A coluna tempo da notificação que consta na tabela 5, consiste no tempo médio do acionamento do sensor até o envio da mensagem para o Websocket. Foi observado que a maior parte deste tempo foi utilizado para criar a imagem da placa, cerca de 6000.0ms. O tempo restante foi utilizado no processo de reconhecimento, por exemplo, 53.2ms para o padrão Mercosul e 45.0ms para o padrão antigo. Também tem o tempo de o envio dos dados para aplicação web, que envolve receber a requisição; buscar na base de dados e enviar a mensagem para o Websocket.

Tabela 5 – Médias dos resultado de testes em ambiente interno.

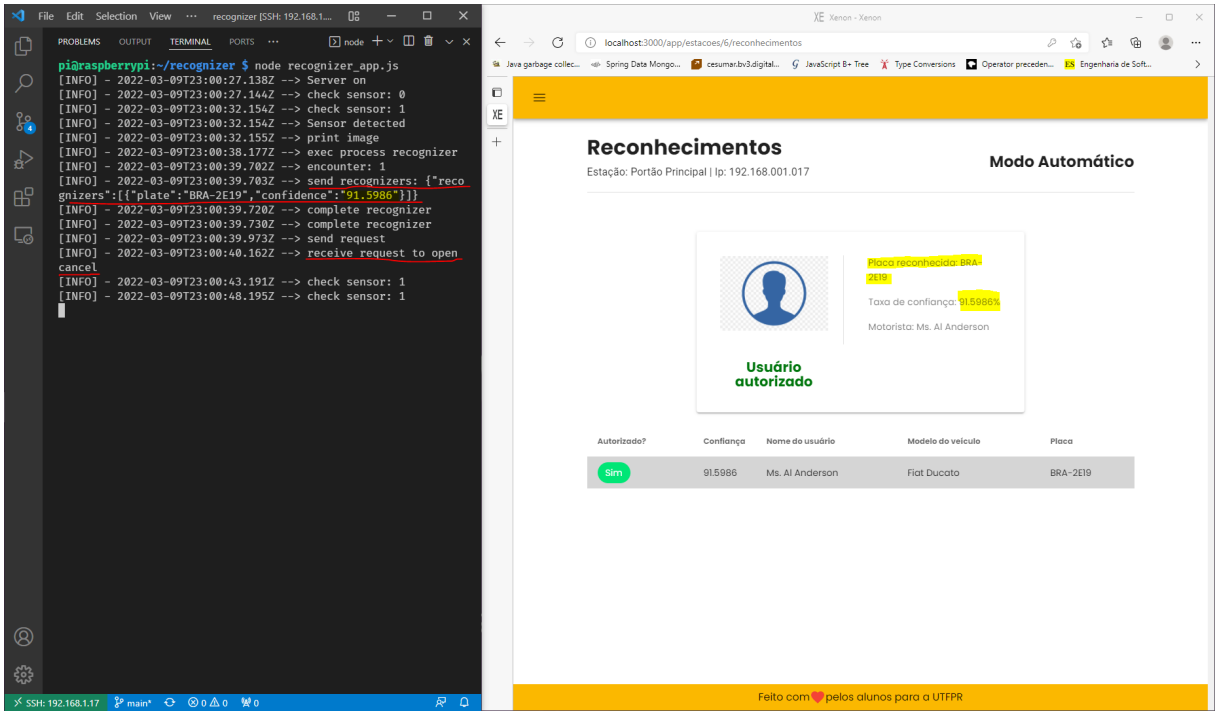
Padrão	Confiabilidade	Tempo Reconhecimento	Tempo da Notificação
Mercosul	82.5889%	53.1870ms	7000.9970ms
Antigo	91.1147%	45.0980ms	7001.0011ms

Fotografia 2 – Exemplo da ação do relé



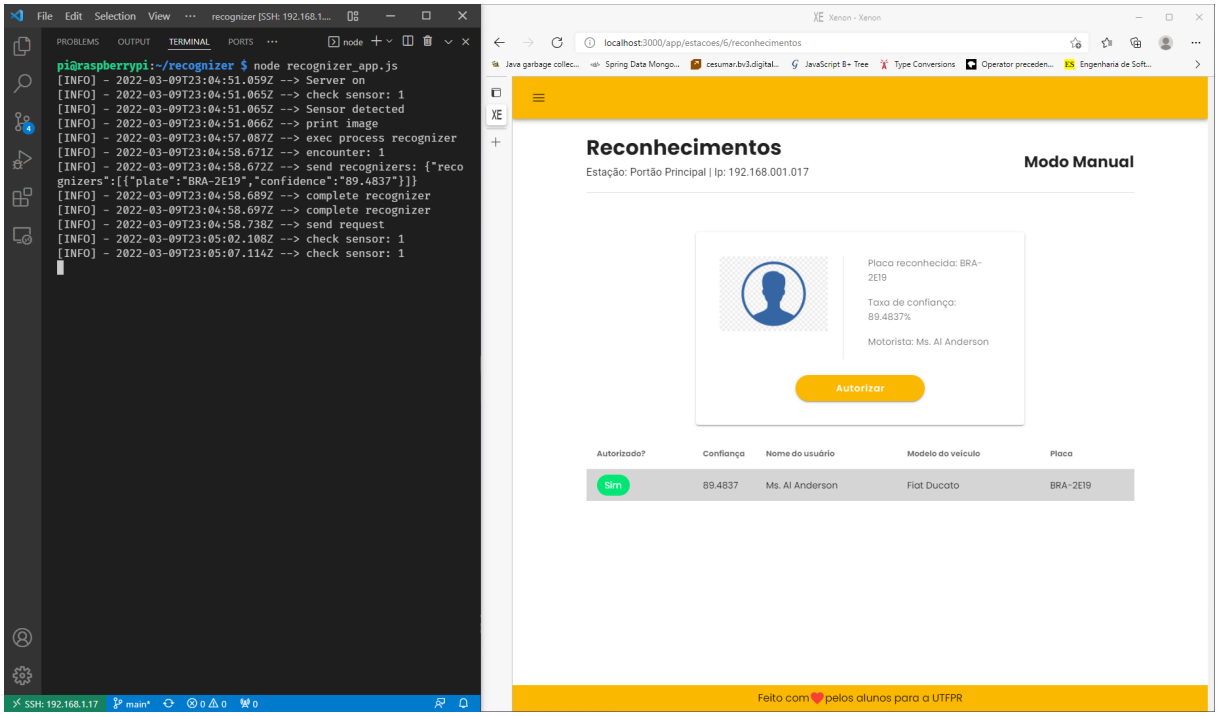
Fonte: autor.

Figura 50 – Teste para estação configurada em modo automático



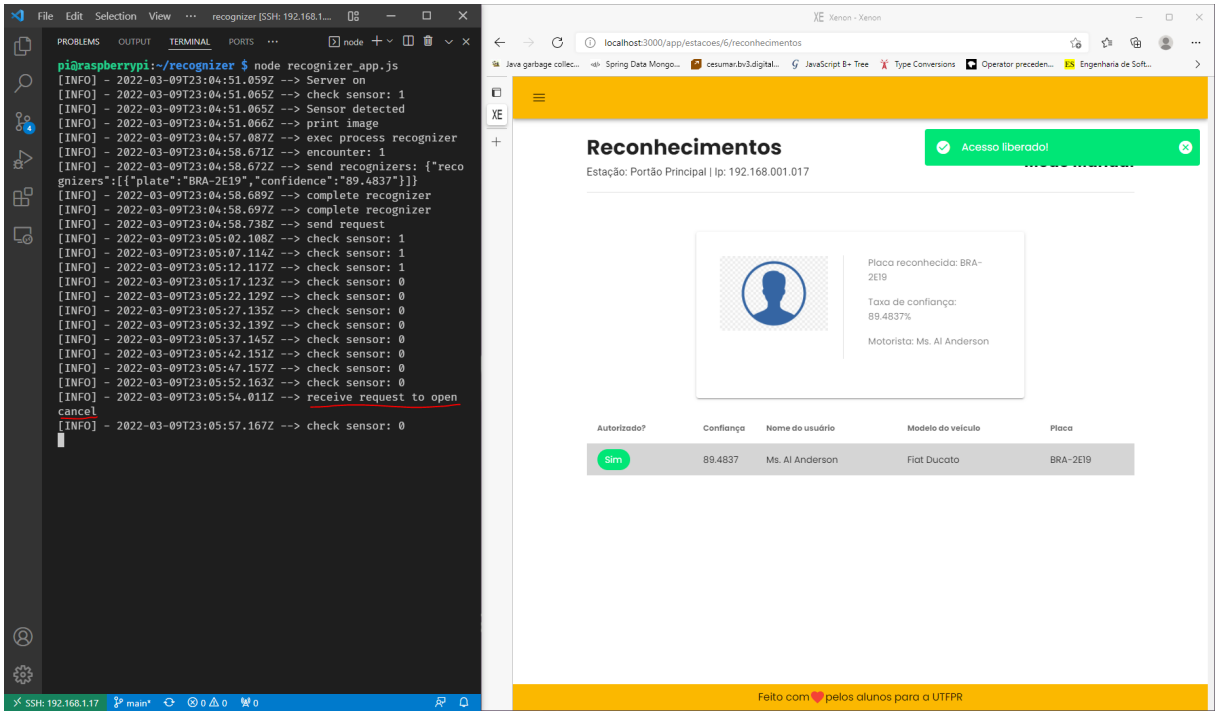
Fonte: autor.

Figura 51 – Teste para estação configurada em modo manual



Fonte: autor.

Figura 52 – Teste para estação configurada em modo manual clicando no botão



Fonte: autor.

7.2 TESTES EXTERNOS

Os testes externos foram feitos em dois horários distintos: um diurno para avaliar o reconhecimento com luz do sol incidindo na câmera, e outro noturno, com a luz do farol do carro incidindo na câmera, sendo que o farol do carro estava em modo baixo.

Para a realização destes testes, o Raspberry foi fixado em uma altura de 50cm a partir do solo. A entrada dos carros foi marcada para que a câmera obtivesse uma imagem em um ângulo aproximadamente de 25 graus à direita do Raspberry em uma distância de 2 metros entre o carro e o Raspberry. Nestes experimentos, foram utilizados 8 carros, sendo 3 com placas no padrão antigo e 5 no padrão Mercosul.

Os resultados do experimento realizado no período diurno é apresentado na tabela 6 e os resultados do experimento realizado no período noturno é apresentado na tabela 7. Em ambos os casos, o algoritmo reconheceu fielmente todas as placas apresentadas. Nestes experimentos, também foram avaliados o nível de confiabilidade na identificação das placas e a média de tempo para a identificação. Mais pontualmente, o valor de confiabilidade consiste o quão certo o reconhecimento pode estar, em geral o algoritmo da biblioteca retorna de 1 a N placas possíveis dessas lista de placas o algoritmo classifica com uma taxa de confiabilidade, isto é, o quão confiante o algoritmo está com as placas reconhecidas, por exemplo. A placa do carro é **AAA-11VC**, devido a alguns fatores, por exemplo ilegibilidade da placa, o algoritmo poderia identificar o carácter 1 como I, dado que para as placas no padrão brasileiro, na quinta posição poderá ser encontrar tanto uma letra com um número. Para esses casos, o algoritmo expões

qual opção ele confia mais. exemplo, na lista de retorno do algoritmo poderá ter ambas as opções, ou seja **AAA-11VC** ou **AAA-1IVC**, mas, para a opção **AAA-11VC** o algoritmo confia que 90% ao passo que **AAA-1IVC** confia apenas 70%, Importante mencionar que mesmo com uma taxa de confiabilidade de 70% o algoritmo conseguiu ao menos extrair da image o conjunto de caracteres que compõe a placa do carro, sendo assim indicando sucesso no reconhecimento. Ademais, em relação ao tempos coletados para criar o imagem até a notificação em tela, foram praticamente os mesmos do ambiente interno, com uma variante apenas de 0.0010ms para mais ou para menos. Por conta disto, nas tabelas de resultados do experimento em ambiente externo, esta coluna foi omitida.

Tabela 6 – Resultado de testes em ambiente externo - diurno.

Padrão	Média de Confiabilidade	Média de Tempo
Mercosul	88.3462%	58.2234ms.
Antigo	94.1385%	54.5431ms.

Tabela 7 – Resultado de testes em ambiente externo - noturno.

Padrão	Média de Confiabilidade	Média de Tempo
Mercosul	78.5444%	64.7113ms.
Antigo	83.9016%	63.1900ms.

Após a análise dos resultados no modo noturno, sendo que tais testes ocorreram com a incidência de luz do farol dos próprios veículos, achou-se necessário aprofundar os testes no período noturno para verificar o comportamento com três(3) variantes: farol desligado, farol baixo e farol alto. Os resultados destes testes estão apresentados respectivamente nas tabelas 8, 9 e 10.

Tabela 8 – Resultado do teste noturno com farol desligado.

Padrão	Média de Confiabilidade	Média de Tempo
Mercosul	83.7765%	53.1870ms.
Antigo	92.0016%	52.0113ms.

Tabela 9 – Resultado do teste noturno com farol no modo baixo.

Padrão	Média de Confiabilidade	Média de Tempo
Mercosul	77.9807%	62.9899ms.
Antigo	84.0009%	64.0097ms.

Tabela 10 – Resultado de teste noturno com farol no modo alto.

Padrão	Média de Confiabilidade	Média de Tempo
Mercosul	Menor que 75%	62.1999ms.
Antigo	Menor que 75%	64.1009ms.

Ao melhor analisar os resultados, entende-se que nos testes com farol desligado os resultados foram mais efetivos pelo fato de não ter incidência de luz direta na câmera do Raspberry, sendo que o mesmo ocorreu com o farol baixo. Assim, o algoritmo de tratamento de imagem do OpenAPLR não obteve problemas em fazer o tratamento da imagem para então extrair os caracteres. O algoritmo reconheceu fielmente todas as placas apresentadas. O mesmo não ocorreu com os testes de farol alto, sendo que algumas placas não foram reconhecidas. As imagens obtidas, como pode ser observado na Figura 3, não contribuíram para o tratamento da imagem. A luz do farol alto incindido diretamente na câmera do Raspberry foi um fator relevante para o OpenALPR apresentar uma taxa de reconhecimentos bem baixa.

Fotografia 3 – Imagem para reconhecimento com Farol Alto

Fonte: autor.

Por fim, as tabelas 11 e 12 apresentam os resultados do reconhecimento para cada automóvel utilizado nos testes com farol alto, respectivamente para as placas no padrão antigo e Mercosul. Cada tabela apresenta o grau de confiabilidade média do algoritmo e se reconheceu ou não o veículo. Com isso, verifica-se que nos testes com farol alto, apenas 2 carros foram bem sucedidos para os padrões antigos, embora com taxa de confiabilidade baixa. Também, apenas 1 dos 5 carros foi reconhecido para o padrão Mercosul, embora também com confiabilidade baixa.

Tabela 11 – Resultado de reconhecimento com farol alto - Padrão Antigo.

Carro	Reconheceu?
Carro 1	Sim
Carro 2	Sim
Carro 3	Não

Enfim, pode-se constatar que a aplicação de reconhecimento foi efetiva tanto nos períodos diurnos e noturnos, desde que o farol esteja desligado ou de modo baixo.

Tabela 12 – Resultado de reconhecimento com farol alto - Padrão Mercosul.

Carro	Reconheceu?
Carro 4	Não.
Carro 5	Não.
Carro 6	Não.
Carro 7	Não.
Carro 8	Sim.

7.3 CONSIDERAÇÕES FINAIS

A aplicação web foi publicada em servidor localizado nos Estados Unidos. A localização do servidor foi proposital, a fim de aumentar a latência de rede e demonstrar o uso da aplicação com uma latência de rede relativamente alta. No total, foram três servidores, sendo que cada servidor possui uma CPU AMD compartilhada com 2 cores disponíveis, 2 GB de memória RAM e 25 GB de espaço de disco SSD. A velocidade da Internet para o Raspberry e o computador do usuário operador foi de aproximadamente 90.7 Mbps/s.

Devido a pandemia vivenciada no período de desenvolvimento deste trabalho, não foi possível fazer a implantação e demonstração no campus da UTFPR. Logo, os experimentos apresentados nas seções anteriores foram focadas na funcionalidade de reconhecimento, pois a quantidade de voluntários foi limitada para não ocorrer aglomeração. Sendo assim, os experimentos não atingiram um quorum alto o suficiente para coletar dados para uma pesquisa de satisfação e usabilidade do sistema.

8 CONCLUSÃO

O resultado final pretendido com o presente trabalho era de entregar o sistema Xenon como uma solução viável para o problema dos acessos ao estacionamento da UTFPR do câmpus Guarapuava, que atualmente ocorre através de adesivos de identificação. Por tudo o que foi apresentado, entende-se que este objetivo foi alcançado, faltando apenas a fase de testes e implantação do sistema no ambiente real. Porém, estes passos independem da vontade do autor deste trabalho, dependendo de decisões administrativas da instituição e principalmente do período pandêmico vivido durante a realização deste trabalho.

Atualmente, o sistema Xenon permite que o departamento responsável pelo gerenciamento do estacionamento da universidade (DERAC) gerencie a liberação dos acessos ao estacionamento, sem onerar os colaboradores. O mesmo benefício é estendido ao corpo de colaboradores da segurança do campus, pois o sistema Xenon não tem a pretensão de substituir a equipe de segurança, mas sim auxiliar em seus trabalhos. O sistema permite que os colaboradores consigam avaliar e auditar os acessos concedidos pelo DERAC. Mesmo que o sistema seja utilizado no modo manual ou quando o sinal de acionamento da cancela está desativado, o colaborador não precisa se preocupar em verificar se o veículo do usuário tem ou não o adesivo, deixando a cargo do sistema que em poucos segundos já informa se o veículo tem ou não autorização para acesso ao estacionamento.

Por sua vez, os discentes têm uma forma simples e fácil de pedir solicitação para o acesso ao estacionamento do campus, facilitando muito o cadastramento e atualização dos dados de seus carros para obter o acesso ao estacionamento.

Também é importante mencionar que o sistema Xenon apresentou algumas falhas, tal como apresentado na seção de resultados. Ele não foi totalmente efetivo quando o automóvel está com farol alto ligado. Porém, em zonas urbanas, raramente o farol alto é ligado.

Um outro ponto é que a implantação do sistema Xenon não é indicado em ambientes que requerem um grande aparato de segurança, por exemplo condomínios residenciais ou empresas que permitem o acesso apenas para funcionários. O seu uso é primordialmente útil para locais públicos, como o próprio campus da Universidade, ou qualquer outro local em que um pedestre pode entrar sem restrições, tal como restaurantes, supermercados e parques. No mesmo sentido, a implantação do sistema é indicado para ambientes em que o estacionamento é gratuito, ou seja, que não requerem pagamentos ou assinaturas. Como o sistema é focado somente na placa do carro para identificação, não há mecanismos de validação de veracidade da placa. A fragilidade na segurança pôde ser constatada no experimento em ambiente interno apresentado no capítulo de resultados, quando uma simples imagem de uma placa exposta em um monitor de TV foi suficiente para ativar a cancela do sistema.

Para estender a aplicabilidade do sistema Xenon, outras formas de reconhecimento podem ser implantadas futuramente. Por exemplo, o sistema poderá suportar acesso via biometria (reconhecimento da digital), reconhecimento facial, QR-Code ou RFID. Porém, na maioria

destas formas adicionais de autenticação, o usuário precisa se expor ao ambiente, por exemplo, abrindo o vidro do carro para o reconhecimento. Além da questão de segurança envolvida nesta exposição, também tem a questão de intempéries, exigindo que haja um gasto adicional para realização de um telhado ou abrigo para o carro enquanto é realizada a autenticação do motorista. Porém, a vantagem da maioria destas formas adicionais, é que elas autenticam o motorista e não o carro. Assim, caso o usuário troque de carro com certa frequência ou seja adepto de aluguel de carros, a sua atividade de ingresso ao ambiente será facilitada. Vale salientar que a forma que mais se aproxima a apresentada neste trabalho, por reconhecimento de placas, é o modo de reconhecimento via RFID. Este possui as mesmas vantagens do modo de reconhecimento de placas, por não ser invasivo, mas apresenta um custo muito maior para a implantação nos veículos. Entretanto, essas novas formas podem complementar o sistema para melhor atender a equipe de segurança na verificação de pedestres.

Um outro ponto a se mencionar é que o sistema Xenon também é extensível para reconhecimentos de placas de motos. O suporte a este tipo de veículo depende apenas da posição da câmera, por exemplo, ser instalada em uma visão traseira da moto, uma vez que as motos não apresentam placas na parte frontal, apenas na traseira.

Por fim, o trabalho abordou diferentes tecnologias, principalmente no contexto de aplicação web, IoT, WoT e Aprendizagem de Máquina. Este conjunto tecnológico vasto permitiu a composição do sistema apresentado, demonstrando como a aplicação de software em hardware pode automatizar processos do cotidiano humano. Ademais, o fato de usar um algoritmo de aprendizagem de máquina da comunidade brasileira, ou seja, contido na biblioteca OpenAPLR, demonstra como a soma do trabalho individual com o trabalho de outras pessoas contribuem positivamente para construir um produto funcional e útil para uma comunidade. Por conta disto, o código-fonte deste trabalho está disponibilizado publicamente, no repositório do GitHub, a fim de incentivar o aprendizado de interessados sobre estas tecnologias utilizadas e principalmente a fim de incentivar iniciativas de trabalhos para evoluírem o sistema a fim de beneficiar um número maior de pessoas.

8.1 TRABALHOS FUTUROS

Um dos trabalhos a ser implementado futuramente está diretamente relacionando com uma maior confiabilidade do acesso. Mais precisamente, achou-se importante incluir uma funcionalidade para notificar a saída do carro do estacionamento. Com esta funcionalidade, seria possível gerenciar a quantidade de vagas ocupadas e disponíveis, pois seria possível calcular entradas e saídas. Esta funcionalidade foi pensada ao final do desenvolvimento, não havendo, portanto, história de usuário para tal.

Também, como trabalhos futuros, seria importante incluir todas as histórias disponíveis no backlog que não foram implementadas nas Sprints realizadas no decorrer deste trabalho. Entre elas, estão as histórias de ID 22 e 23, como apresentado na Tabela 13, que em conjunto

com a funcionalidade apresentada anteriormente, pode melhorar a usabilidade do sistema Xenon. Além destas, um trabalho futuro significativo seria no sentido de incluir o suporte a novas formas de autenticação de usuários motoristas ou veículos, ou seja, via biometria, reconhecimento facial, QR-Code impresso ou gerado dinamicamente no aplicativo de celular e, mesmo a tecnologia de RFID.

Em consoante, como observado na analise dos testes, no capitulo **??**, que o tempo total do reconhecimento, ou seja, do momento que o carro entra no campo de visão do sensor até a notificação para o usuário, foi de aproximadamente seis segundos, além de, que a maioria deste tempo foi gasto no processo de tirar a foto. Este tempo embora satisfatória pode ser melhorado, um estudo com o foco de melhorar este tempo, propondo novas abordagens, ou equipamentos afim diminuir esse tempo.

Tabela 13 – Histórias que não foram concluídas.

ID	Prioridade	História
22	1	COMO administrador do sistema QUERO que os cadastros de alunos sejam desativado com o aluno após a conclusão do curso.
23	1	COMO administrador do sistema QUERO quero cadastrar um usuário motorista com e configurar um tempo para a conta ficar ativa.

REFERÊNCIAS

ASHTON, K. **That 'Internet of Things' Thing**. 2009. Disponível em: https://www.rfidjournal.com/that-internet-of-things-thing. Acesso em: 21 de abril de 2021.

ATLASSIAN, E. **Fluxo de trabalho de Gitflow**. [*S.l.*], 2021. Disponível em: https://www.atlassian.com/br/git/tutorials/comparing-workflows/gitflow-workflow. Acesso em: 21 de abril de 2021.

CAM, E. R. **Módulo Câmera**: Documentação módulo câmera. [*S.l.*], 2021. Disponível em: https://projects.raspberrypi.org/en/projects/getting-started-with-picamera. Acesso em: 21 de abril de 2021.

CODECOV. **Codecov – Documentantion**. 2021. Disponível em: https://docs.codecov.com/docs. Acesso em: 7 de outubro de 2020.

DBA. **dba.eng.br**. 2021. Disponível em: https://dba.eng.br/br/solucoes/ocr/ocr-alpr. Acesso em: 21 de abril de 2021.

DOCS, M. W. **Introdução Express/Node**. 2021. Disponível em: https://www.infoescola.com/informatica/cascading-style-sheets-css/. Acesso em: 01 de dezembro de 2021.

FIELDING, R. T.; TAYLOR, R. N. **Architectural styles and the design of network-based software architectures**. [*S.l.*]: University of California, Irvine Doctoral dissertation, 2000. v. 7.

FILIPEFLOP, E. **Módulo Sensor de Presença**: Documentação módulo câmera. [*S.l.*], 2021. Disponível em: https://www.filipeflop.com/produto/sensor-de-movimento-presenca-pir/. Acesso em: 21 de abril de 2021.

FOWLER, M. **Padrões de Arquitetura de Aplicações Corporativas**. [*S.l.*]: Bookman, 2009. ISBN 9788577800643.

FRAMEWORK, E. S. **SETIC, SCRUM**: Coordenação de gestão estratégico. [*S.l.*], 2022. Disponível em: https://spring.io/. Acesso em: 21 de abril de 2021.

FRAMEWORK, E. S. **Spring Framework**: Documentação spring framework. [*S.l.*], 2022. Disponível em: https://spring.io/. Acesso em: 21 de abril de 2021.

FRANÇA, T. C. de *et al.* **Web das coisas: conectando dispositivos físicos ao mundo digital**. 2011.

GENETEC. **genetec.com**. 2021. Disponível em: https://www.genetec.com/br/solucoes/todos-os-produtos/autovu. Acesso em: 21 de abril de 2021.

GIT. **Git Documentation**. 2005. Disponível em: https://git-scm.com/docs/git. Acesso em: 21 de abril de 2021.

GPIO, E. R. **Documentação Raspberry**: Documentação gpio. [*S.l.*], 2021. Disponível em: https://www.raspberrypi.org/documentation/usage/gpio/. Acesso em: 21 de abril de 2021.

IBM. **Arquitetura de três camadas (tiers)**. 2020. Disponível em: https://www.ibm.com/br-pt/cloud/learn/three-tier-architecture. Acesso em: 21 de abril de 2021.

JACKSON, L. **OV5647 Mini Camera Module for Raspberry Pi**: Coordenação de gestão estratégico. [*S.l.*], 2015. Disponível em: https://www.arducam.com/lowcost-raspberry-pi-mini-camera-module/. Acesso em: 21 de abril de 2021.

JOHNSON, R. *et al.* The spring framework–reference documentation. **interface**, v. 21, p. 27, 2004.

LUCKOW, D. H.; MELO, A. A. de. **Programação Java para a WEB**. [*S.l.*]: Novatec Editora, 2010.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. **Revista de Informática Teórica e Aplicada**, v. 16, n. 1, p. 125–160, 2009.

MEERKAT. **Meerkat.com/alpr**. 2021. Disponível em: https://www.meerkat.com.br/solution_automatic_license_plate_recognition.html. Acesso em: 21 de abril de 2021.

MICROSOFT.DOCS. **Mapeamentos de PIN do Raspberry Pi 2 3**. 2021. Disponível em: https://docs.microsoft.com/pt-br/windows/iot-core/learn-about-hardware/pinmappings/pinmappingsrpi. Acesso em: 7 de outubro de 2020.

MILANI, A. Postgresql - guia do programador. **interface**, v. 1, p. 392, 2008.

OKDO. **5MP Camera User Manual**. 2021. Disponível em: ttps://docs.rs-online.com/b064/A700000006917308.pdf. Acesso em: 01 de dezembro de 2021.

OTTO, M. **Bootstrap from Twitter**. 2011. Disponível em: https://www.infoescola.com/informatica/cascading-style-sheets-css/. Acesso em: 01 de dezembro de 2021.

PACíEVITCH, Y. **Cascading Style Sheets (CSS)**. 2021. Disponível em: https://www.infoescola.com/informatica/cascading-style-sheets-css/. Acesso em: 01 de dezembro de 2021.

RUSSEL STUART; NORVIG, P. **Inteligência Artificial**. 2. ed. Rio de Janeiro: Campos, 2013.

SABBAGH, R. **Gestão Ágil para projetos de sucesso**. São Paulo: Casa do Código, 2013.

SANFILIPPO, S. **The End of the Redis Adventure**. 2020. Disponível em: http://antirez.com/news/133. Acesso em: 21 de abril de 2021.

SCURI, A. E. Fundamentos da imagem digital. **Pontifícia Universidade Católica do Rio de Janeiro**, 1999.

SUTHERLAND, K. S. e. J. **Guia do Scrum**. 2013. Disponível em: https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf. Acesso em: 21 de abril de 2021.

TAKATUZI, F. K. O. **Adaptive-IDT: algoritmo incremental para aprendizagem de árvores de decisão adaptativas**. 2017. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2017.

TEAM, T. T. **Thymeleaf**. 2013. Disponível em: https://www.thymeleaf.org/documentation.html. Acesso em: 21 de abril de 2021.

TECHNOLOGY, I. O. **OpenALPR Documentation**: Getting started. [*S.l.*], 2017. Disponível em: http://http://doc.openalpr.com/. Acesso em: 12 de maio de 2021.

TRAVIS. **Travis-CI - Documentation**. 2005. Disponível em: https://docs.travis-ci.com/. Acesso em: 21 de abril de 2021.

TYPESCRIPTLANG.ORG. **TypeScript for JavaScript Programmers**. 2021. Disponível em: https://www.typescriptlang.org/. Acesso em: 21 de abril de 2021.

VERNON, V. **Implementando o Domain-Driven Design**. [*S.l.*]: Alta Book, 2016. v. 1.

VUEJS.ORG. **The Progressive JavaScript Framework**. 2014. Disponível em: https://vuejs.org/. Acesso em: 21 de abril de 2021.

W3C. **HTML E CSS**. 2016. Disponível em: https://www.w3.org/standards/webdesign/htmlcss. Acesso em: 21 de abril de 2021.

WHATWG. **Living Standard**. 2021. Disponível em: https://html.spec.whatwg.org/multipage/. Acesso em: 01 de dezembro de 2021.

ZENHUB. **ZenHub – Project Management Inside GitHub**. 2020. Disponível em: https://chrome.google.com/webstore/detail/zenhub-for-github/ogcgkffhplmphkaahpmffcafajaocjbd?hl=pt-BR. Acesso em: 7 de outubro de 2020.