UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MARCOS CHAVES PAIM

APRIMORAMENTO DE UM SISTEMA DE INFORMAÇÃO AO PASSAGEIRO PARA ESTAÇÕES METROVIÁRIAS

Porto Alegre
2024

MARCOS CHAVES PAIM

APRIMORAMENTO DE UM SISTEMA DE INFORMAÇÃO AO PASSAGEIRO PARA ESTAÇÕES METROVIÁRIAS

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Engenheiro Eletricista.

ORIENTADOR: Prof. Dr. Paulo Francisco Butzen

Porto Alegre
2024

MARCOS CHAVES PAIM

APRIMORAMENTO DE UM SISTEMA DE INFORMAÇÃO AO PASSAGEIRO PARA ESTAÇÕES METROVIÁRIAS

Este Projeto foi julgado adequado para a obtenção dos créditos da Disciplina Projeto de Diplomação do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Paulo Francisco Butzen,

Banca Examinadora:

Prof. Dr. Raphael Martins Brum, UFRGS

Prof. Dr. Tiago Oliveira Weber, UFRGS

Eng. David Samuel Levenfus, Trensurb

Coordenador do Curso: _____
Prof. Dr. Ivan Müller

Porto Alegre, fevereiro de 2024.

RESUMO

Sistemas de informação ao passageiro são importantes para comunicar aos usuários do transporte público informações operacionais, como a previsão do tempo de chegada dos trens, mensagens de alerta e mudanças de plataforma. Serviços como esse aprimoram a experiência no transporte público, tornando-o mais confortável e acessível, e promovem sua maior utilização ao torná-lo uma escolha mais atrativa para os usuários. O presente trabalho tem o objetivo de aprimorar sistema de informação ao usuário destinado às estações da Trensub. As principais melhorias e novas funcionalidades propostas são a implementação de páginas de administração e de gerenciamento de mensagens de alerta na aplicação Web que compõe o sistema, a criação de um novo módulo cliente utilizando uma *Smart TV* e realizando o desenvolvimento de um aplicativo *Android* personalizado e a criação de modelos de previsão do tempo de chegada dos trens nas estações utilizando técnicas de aprendizado de máquina, juntamente com uma refatoração do software da aplicação Web para comportar essas melhorias. Para a construção da aplicação Web utilizou-se o *framework Django* da linguagem *Python 3*, e o banco de dados *MySQL*. Para a criação dos modelos de predição, utilizaram-se as técnicas *k-nearest neighbours*, rede neural MLP e Floresta Aleatória. Também foi construído um algoritmo analítico para calcular a previsão a partir da velocidade média de operação. Um conjunto de dados com informações de geolocalização dos trens foi criado para realizar o treinamento e a validação dos modelos. A métrica de erro utilizada para avaliar os resultados foi o *Root Mean Squared Error* e demonstrou-se que os modelos de Floresta Aleatória apresentaram o menor erro com o conjunto teste. Por fim, os modelos treinado foram integrados à aplicação e o sistema final demonstrou atender aos requisitos levantados.

Palavras-chave: Sistema de Informação a Passageiros, Painéis Informativos, Trem urbano, Aprendizado de Máquina.

ABSTRACT

Passenger information systems are important for communicating operational information to public transport users, such as the estimated time of arrival of trains, alert messages, and platform changes. Services like these enhance the public transport experience, making it more comfortable and accessible, and promote its increased use by making it a more attractive choice for users. The present work aims to improve the user information system intended for Trensub stations. The main improvements and new functionalities proposed are the implementation of administration pages and alert message management in the Web application that makes up the system, the creation of a new client module using a *Smart TV*, the development of a custom *Android* app, and the development of train arrival time prediction models at stations using machine learning techniques, along with a refactoring of the Web application software to accommodate these improvements. The *Django framework* of the *Python 3* language, and the *MySQL* database were used for the Web application development. For the creation of the prediction models, the *k-nearest neighbours*, MLP neural network, and Random Forest techniques were utilized. An analytical algorithm was also constructed to calculate the prediction from the average operational speed. A dataset with train geolocation information was created to perform the training and validation of the models. The error metric used to evaluate the results was the *Root Mean Squared Error*, and it was demonstrated that the Random Forest models presented the lowest error with the test set. Finally, the trained models were integrated into the application, and the final system was shown to meet its requirements.

Keywords: Passenger Information System, Information Panels, Urban Train, Machine Learning.

LISTA DE ILUSTRAÇÕES

Figura 1: Painel de informação 14
Figura 2: Exemplo de árvore de decisão 17
Figura 3: Rede Neural *Feed-forward* 18
Figura 4: Comunicação entre clientes e servidor no HTTP 20
Figura 5: Visão geral do sistema proposto 24
Figura 6: Tela de horário 25
Figura 7: Tela de mídia com imagem de avisos institucionais 25
Figura 8: Raspberry Pi 26
Figura 9: Cliente do prédio administrativo 27
Figura 10: GUI desenvolvida 28
Figura 11: Lógica de requisição e resposta 31
Figura 12: Diagrama do modelo entidade relacionamento 32
Figura 13: Fluxograma da metodologia de criação dos modelos 35
Figura 14: Diagrama de entidade e relacionamento do banco de dados coletado . 37
Figura 15: Fluxograma de previsão 40
Figura 16: Arquitetura do novo sistema 43
Figura 17: Diagrama MTV 44
Figura 18: Tela de horário 45
Figura 19: Tela de previsão 45
Figura 20: Tela com trens no trecho de via 45
Figura 21: Tela de imagem institucional 46
Figura 22: Relacionamento e atributos 47
Figura 23: Tela de login 47
Figura 24: Interface administrativa 48
Figura 25: Bloco de Programação 48
Figura 26: Interface de mensagens de alerta 49
Figura 27: Adicionar alerta 50
Figura 28: Quadro de mensagens 51
Figura 29: Exemplo de página com mensagem de alerta 51
Figura 30: Tabela de programação do exemplo 52
Figura 31: Exemplo de sequência de programação 52
Figura 32: Exemplo de sequência de programação com mensagem de alerta . . . 53
Figura 33: Tela de previsão na *Smart TV* 54
Figura 34: Menu de seleção de IP e porta 54
Figura 35: Menu de seleção de estação 55
Figura 36: Menu de seleção de local 55

LISTA DE TABELAS

Tabela 1: Parâmetros do sistema TETRA 15
Tabela 2: Número de amostras do conjunto de dados de cada estação 37
Tabela 3: Valores dos Hiperparâmetros 39
Tabela 4: Características escolhidas 56
Tabela 5: Tabela horária de partidas da estação Mercado em dias úteis 57
Tabela 6: Tabela horária de partidas da estação Mercado nos sábados 57
Tabela 7: Tabela horária de partidas da estação Mercado nos domingos e feriados 58
Tabela 8: Hiperparâmetros dos modelos para cada estação 59
Tabela 9: Raiz do erro médio quadrático dos modelos e algoritmo analítico . . . 60
Tabela 10: Tamanho dos modelos em KB 61

LISTA DE ABREVIATURAS

API	Interface de Programação de Aplicativos (*Application Programming Interface*)
APK	*Android Application Pack*
CCO	Centro de Controle Operacional
CSS	Folhas de Estilo em Cascata (*Cascading Style Sheets*)
GECIN	Gerência de Comunicação Integrada
GPS	Sistema de posicionamento global (*Global Positioning System*)
GSM-R	*Global System for Mobile Communications – Railway*
HDMI	Interface Multimídia de Alta Definição (*High-Definition Multimedia Interface*)
HTML	Linguagem de Marcação de Hipertexto (*HyperText Markup Language*)
HTTP	Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*)
IDE	Ambiente de Desenvolvimento Integrado (*Integrated Development Environment*)
IP	Protocolo de Internet (*Internet Protocol*)
MLP	*Multilayer Perceptron*
MTV	*Model-Template-View*
MVC	*Model-View-Controller*
NTP	Protocolo de Tempo para Redes (*Network Time Protocol*)
ORM	Mapeamento objeto-relacional (*Object-relational mapping*)
RMSE	Raiz do Erro Quadrático Médio (*root-mean-square error*)
SMS	Serviço de mensagens curtas (*Short Message Service*)
SQL	*Structured Query Language*
TCP	Protocolo de Controle de Transmissão (*Transmission Control Protocol*)
TETRA	Rádio Troncalizado Terrestre (*Terrestrial Trunked Radio Access*)
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
VLAN	Rede Local Virtual (*Virtual Local Area Network*)
VNC	Computação em Rede Virtual (*Virtual Network Computing*)

SUMÁRIO

1 INTRODUÇÃO 10

2 REFERENCIAL TEÓRICO 13
2.1 Sistemas de Informação ao Passageiro 13
2.2 Terrestrial Trunked Radio Access - TETRA 14
2.3 Aprendizado de máquina 15
2.3.1 K-Nearest Neighbours 16
2.3.2 Árvores de Decisão 16
2.3.3 Floresta Aleatória 17
2.3.4 Redes Neurais Artificiais 17
2.4 Arquitetura TCP/IP 18
2.5 HTTP 19
2.5.1 Mensagem de requisição 20
2.5.2 Mensagem de resposta 21

3 SISTEMA EXISTENTE 23
3.1 Visão Geral do Sistema 23
3.2 Módulo Servidor 23
3.3 Módulo Cliente 24
3.4 Módulo de Administração 26

4 METODOLOGIA 29
4.1 Atualização do módulo servidor 29
4.1.1 Rotas e Lógica de Requisição e Resposta 30
4.1.2 Banco de dados 31
4.1.3 Novo módulo de administração 33
4.1.4 Sistema de Mensagens de Alerta 33
4.2 Novo módulo cliente 34
4.3 Criação e Implementação do modelo de previsão para informar o tempo de chegada dos trens 34
4.3.1 Coleta da Base de Dados 35
4.3.2 Processamento dos dados 36
4.3.3 Análise das melhores características 38
4.3.4 Otimização dos hiperparâmetros e Análise dos modelos 39
4.3.5 Implementação dos modelos na aplicação 40

5 RESULTADOS 42
5.1 Atualização do Módulo Servidor 42
5.1.1 Código 43
5.1.2 Rotas e Lógica de Requisição e Resposta 44
5.1.3 Novo módulo de administração 46
5.1.4 Sistema de Mensagens de Alerta 49
5.1.5 Exemplo de requisição 51
5.2 Novo módulo cliente 53
5.3 Modelo de previsão para informar o tempo de chegada dos trens 56
5.3.1 Escolha das melhores características 56
5.3.2 Otimização dos hiperparâmetros e treinamento dos modelos 58

6 CONCLUSÃO 62

REFERÊNCIAS 64

1 INTRODUÇÃO

O transporte público desempenha um papel fundamental na sociedade moderna, pois auxilia no desenvolvimento sustentável e no aumento da qualidade de vida nas cidades. Ao oferecer uma alternativa viável ao transporte particular, o sistema de transporte público reduz o congestionamento nas vias, diminui a poluição do ar e atenua os impactos negativos do uso excessivo de veículos privados. Dentre os princípios nos quais a Política Nacional de Mobilidade Urbana está fundamentada, encontram-se a eficiência, eficácia e efetividade na prestação dos serviços de transporte urbano e na circulação urbana. Para cumprir esses objetivos, é essencial promover o aprimoramento do transporte público.

Segundo Caulfield e O'Mahony (2007), a maioria dos usuários de transporte público sentem-se frustrados com a incerteza em relação ao tempo de chegada do serviço e painéis de informação ao passageiro em tempo real é o método mais popular de obtenção de informações. Sendo assim, a implementação de sistemas de informação ao passageiro no transporte público pode melhorar a experiência dos usuários e também atrair mais pessoas ao sistema.

A Empresa de Trens Urbanos de Porto Alegre S/A – Trensurb é uma empresa pública de capital fechado, criada em 1980 para implantar e operar o serviço de trens urbanos na Região Metropolitana de Porto Alegre. Atualmente opera uma linha com extensão de 43,8 quilômetros e 23 estações, atendendo aos municípios de Porto Alegre, Canoas, Esteio, Sapucaia do Sul, São Leopoldo e Novo Hamburgo e um sistema de Aeromóvel com duas estações de embarque e dois veículos, permitindo a integração e o rápido acesso ao Aeroporto Internacional Salgado Filho em Porto Alegre. Dentre os processos de sustentação da empresa, está o desenvolvimento tecnológico, que se refere ao desenvolvimento de novos produtos e processos, melhorando o desempenho operacional, a infraestrutura, a prestação de serviços ao cliente e os controles. Em 2022, a empresa transportou em média 107.742 passageiros por dia (TRENSURB, 2023).

Considerando o sistema metroviário descrito e as funcionalidades e benefícios que a implementação de sistemas de informação ao usuário podem proporcionar ao transporte público, é de interesse da empresa que o metrô da região metropolitana de Porto Alegre seja contemplado com um sistema desse tipo, com o objetivo principal de implementar mais um canal de comunicação com o usuário a fim de transmitir informações importantes sobre a operação do metrô. Este sistema deve ser flexível, escalável e deve se adequar à infraestrutura existente da empresa.

Em (STERNBERG, 2021), é proposto um sistema de informações ao usuário para as estações da Trensurb. Este sistema é baseado em uma arquitetura cliente-servidor, em que painéis anunciadores localizados nas estações comunicam-se com um servidor central e apresentam conteúdos definidos por um programa de administração.

Este projeto foi organizado em três módulos: o módulo cliente, o módulo servidor e o

módulo administrativo. O módulo cliente é composto por *Raspberrys Pi* conectados a televisores que se comunicam com o módulo servidor a partir de uma rede vlan da empresa. O módulo servidor atua como um servidor HTTP (do inglês, *Hypertext Transfer Protocol*), que responde às requisições enviadas pelos clientes e envia conteúdos específicos para cada estação. Por sua vez, o módulo administrativo é composto por uma aplicação Desktop com uma interface gráfica onde é mostrado a condição de conexão dos clientes e podem ser escolhidas imagens ou vídeos para serem exibidas nos clientes de cada estação. O projeto foi desenvolvido com cinco principais serviços:

- O serviço de apresentação de informação aos painéis anunciadores de informação, com o desenvolvimento de uma interface gráfica para a atualização das informações, e de painéis para apresentar as informações seguindo a arquitetura cliente-servidor;
- O serviço de teste de conexão, realizado com um PING periódico do servidor para os clientes;
- O serviço de servidor local de backup, entrando em vigor em caso de perda de conexão entre o cliente e o servidor, não entrando em funcionamento em caso de normalidade de conexão;
- O serviço de sincronização horária das telas, feita com um servidor Protocolo de Tempo para Redes (NTP);
- O serviço de acesso remoto dos painéis, usado para visualização, atualização de arquivos e de software de maneira remota.

Analisando a implementação do trabalho anterior a partir de novos estudos, é possível identificar algumas deficiências, além dos pontos em abertos citados pelo autor. As principais lacunas e novas funcionalidades que este trabalho pretende abordar são:

- A necessidade de adaptar o sistema para suportar múltiplos clientes em cada estação, possibilitando a exibição de conteúdos distintos para diferentes locais de uma mesma estação. Isso demanda extensivas modificações no software dos módulos administrativo e servidor, uma vez que o *design* original previa apenas um painel por estação, sem capacidade para gerenciar múltiplos painéis com conteúdos variados.
- Informar o tempo de chegada do trem para a plataforma, por meio de um modelo de previsão e da criação de telas para serem exibidas nas estações. Essa funcionalidade exige, primeiramente, desenvolver um sistema que capte a localização dos trens em tempo real e faça a previsão do tempo de chegada desses trens à determinada estação. Será feito um estudo da utilização de um algoritmo analítico e também a utilização de diferentes técnicas de aprendizado de máquina supervisionado para o modelo de previsão.
- A substituição dos *Raspberrys Pi* conectados a televisões por *Smart Tvs* com um aplicativo personalizado instalado. A importância dessa mudança se deve a falhas apresentadas pelos *Raspberrys Pi* no sistema original. Com essa mudança, espera-se uma redução na incidência de falhas técnicas, além de oferecer vantagens econômicas, tornando o sistema mais econômico.

- A migração do módulo de administração para uma interface baseada em página Web, hospedada pelo servidor central, facilitará o gerenciamento do sistema por diferentes setores e em variados locais da empresa. Essa mudança é fundamental para superar as limitações do sistema inicial, que restringia o acesso administrativo a um único computador, onde o servidor estava instalado.

- A criação e automação de um modelo para a apresentação prioritária de alertas pelo Centro de Controle Operacional (CCO) a fim de melhorar a comunicação com os passageiros, especialmente em situações não rotineiras. Esta funcionalidade é essencial para manter os usuários bem informados e seguros durante a operação do metrô.

Este trabalho tem como objetivo dar continuidade ao projeto descrito anteriormente, implementando melhorias sugeridas no trabalho anterior e também novas funcionalidades propostas neste trabalho. O aprimoramento do projeto será feito principalmente por meio de mudanças no software do sistema e de mudanças no hardware para adequá-lo aos novos recursos. Para o modelo de previsão, será criada uma base de dados, a qual será analisada e usada para o treinamento de modelos de previsão de chegada dos trens.

Este trabalho está estruturado da seguinte forma. No capítulo 2 é apresentada uma revisão bibliográfica sobre os conceitos abordados no trabalho. No capítulo 3 realiza-se uma revisão do sistema ao qual este trabalho propõe melhorias. O capítulo 4 detalha as tecnologias e os métodos utilizados para a implementação das novas funcionalidades. O capítulo 5 apresenta uma demonstração do novo sistema e os resultados da avaliação dos modelos criados. Por fim, o capítulo 6 apresenta conclusão e os trabalhos futuros relacionados a este projeto.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão apresentados os principais conceitos relacionados à elaboração deste trabalho. Serão abordados os fundamentos sobre sistemas de informação ao passageiro no transporte público, técnicas de aprendizado de máquina e os protocolos de comunicação utilizados para o desenvolvimento do sistema.

2.1 Sistemas de Informação ao Passageiro

Sistemas de informação ao passageiro são um conjunto de ferramentas e tecnologias aplicadas para que o operador de um sistema de transporte público transmita informações aos usuários. Essas informações podem ser a previsão de chegada de veículos, mensagens sobre a situação operacional, preço da tarifa de transporte, entre outras. Os meios mais comuns de apresentação de informação aos usuários de transporte público são: tabelas de horários, mapas de rotas, máquinas de venda de bilhetes, painéis de informação, serviços telefônicos e serviços baseados na Web (SKOGLUND, 2012). Caulfield e O'Mahony (2007) definiram alguns tipos de sistemas de informação ao passageiro:

- Internet: *Websites* da empresa operadora de transporte publico, que pode conter informações operacionais, tabela de horário, posição dos veículos em tempo real e demais informações úteis ao usuário.
- Painéis de informação em tempo real: Televisores que informam o tempo estimado de chegada de veículos e outras informações operacionais, instalados nas estações ou paradas.
- Tecnologia móvel de informações: Envio de informações a telefones celulares de passageiros por meio de SMS.
- Quiosques de informação: Estruturas, com ou sem funcionários, que fornecem aos usuários informações sobre o transporte publico.
- Cartazes de informação: Cartazes com tabelas de horários, disponíveis nas estações ou diretamente com a empresa.

A Figura 1 mostra um exemplo de painel de informação instalado na estação Palmeiras-Barra Fundado do metrô em São Paulo, que exibe a previsão de chegada do próximo trem em tempo real e o nível de ocupação de cada um de seus vagões. A primeira fase desse projeto, que prevê a instalação de equipamentos em duas estações, terá um investimento de 14,7 milhões de reais. Os painéis que estarão nas plataformas mostrarão o tempo estimado de chegada do próximo trem, informações sobre a operação das linhas, destino do

trem e poderá ser feita inclusão de outras informações operacionais. Segundo a empresa, as informações serão atualizadas de forma automática, em tempo real, e com comunicação direta com o Centro de Controle Operacional (LOBO, 2024).

Figura 1: Painel de informação

Fonte: (LOBO, 2024)

Um estudo conduzido por Caulfield e O'Mahony (2007) examinou a visão dos passageiros sobre esses sistemas de informação ao passageiro do transporte público em Dublin, na Irlanda. O estudo mostrou que os usuários do transporte público de Dublin gostariam que houvesse melhorias nos sistemas de informação ao passageiro e que o fornecimento de informações em tempo real foi considerado o método mais importante de fornecimento de informações. Entre os tipos de sistemas de informação ao passageiro, painéis de informações foram o tipo mais popular entre os usuários, seguido por SMS (serviço de mensagens curtas, do inglês *Short Message Service*).

2.2 Terrestrial Trunked Radio Access - TETRA

A operação de uma rede ferroviária exige a integração de múltiplos subsistemas, como a infraestrutura da linha ferroviária, o sistema de energia, o gerenciamento e operação do tráfego, entre outros. Os sistemas de sinalização e de redes de comunicações são dois subsistemas essenciais para as redes ferroviárias. O primeiro incluí todos os equipamentos necessários para garantir a segurança e o controle do movimento dos trens. Já o segundo consiste na transmissão de dados para o sistema de sinalização e a comunicação de voz para os usuários do sistema, como operadores, passageiros e equipes de manutenção.

O sistema TETRA propõe a otimização da integração entre esses dois sistemas, com uma solução economicamente eficiente, flexível para aperfeiçoamentos, assim garantindo a manutenção e o aprimoramento do sistema. Ele é uma alternativa mais moderna, que vem sendo cada vez mais adotada, ao sistema GSM-R (do inglês, *Global System for Mobile Communications – Railway*).

O sistema TETRA opera entre as bandas de frequência de 300 Mhz e 800 MHz, en-

quanto o GSM-R opera entre 900 Mhz e 1800 Mhz. Devido a isso, o GSM-R apresenta maiores perdas de propagação, necessitando de mais estações de repetição em comparação com o TETRA. O sistema TETRA, portanto, tem menores custos de implantação. Além disso, ele possui uma maior frequência espectral, pois tem 4 canais em uma banda de 25 Khz, enquanto o sistema GSM-R apresenta 8 canais em uma banda de 200Khz. Os principais parâmetros do sistema TETRA são mostrados na tabela 1.

Tabela 1: Parâmetros do sistema TETRA

Parâmetro	Valor
Espaçamento entre portadoras	25 kHz
Modulação	$\pi/4$ DQPSK
Taxa de Transmissão da portadora	36 kbit/s
Taxa de codificação	ACELP
Acesso	TDMA
Taxa de Transmissão de Usuário	7,2kbits/s por time slot
Taxa Máxima de Transmissão	28,8 kbit/s
taxa de dados protegida	até 19,2 kbit/s

Fonte: DUNLOP; GIRMA; IRVINE, 1999

2.3 Aprendizado de máquina

O aprendizado de máquina (em inglês, *machine learning*) é um conjunto de técnicas que envolvem o uso de um computador para observar um conjunto de dados, criar um modelo baseado nesses dados e usar tal modelo tanto como uma hipótese sobre algum fenômeno quanto como um software cujo objetivo é resolver determinado problema. Para isso, são utilizadas técnicas estatísticas fundamentais para inferir estruturas e padrões, permitindo que algoritmos aprendam a partir de conjuntos de dados e melhorem suas previsões ou decisões com base em novas informações. Quando a saída de um modelo de aprendizado de máquina é um conjunto finito de valores, por exemplo, chuvoso, ensolarado e nublado, trata-se de um problema de classificação. Por outro lado, quando a saída do modelo é um número, por exemplo, temperatura, trata-se de um problema de regressão (RUSSELL; NORVIG, 2020).

Os algoritmos de aprendizado de máquina podem ser divididos segundo o seu processo de aprendizado. De acordo com Russell e Norvig (2020), existem três tipos de aprendizado: supervisionado, não supervisionado e aprendizado por reforço.

No aprendizado supervisionado são utilizados conjuntos de amostras com entradas e saídas conhecidas, e, a partir desses dados, um modelo é treinado para prever com precisão a saída de observações futuras. Por exemplo, considerando um veículo autônomo que utiliza modelos treinados a partir do comportamento de um condutor humano, um modelo de decisão de quando treinar pode ter como entradas a velocidade e a direção atuais do veículo e as condições da estrada, e como saída a distância de frenagem.

No aprendizado não supervisionado, as amostras do conjunto de dados não possuem saídas rotuladas, e os algoritmos devem encontrar padrões ou estruturas nos dados. A técnica mais comum de aprendizado não supervisionado é o *clustering*, que envolve detectar potenciais agrupamentos no conjunto de dados. Um exemplo de aplicação desta técnica

é, dado um conjunto de milhares de imagens retiradas da internet, um sistema de visão computacional consegue identificar um *cluster* de imagens que seriam reconhecidas como "gatos"por observadores humanos (RUSSELL; NORVIG, 2020).

No aprendizado por reforço, um modelo é treinado a partir de punições e recompensas e o algoritmo decide, após cada iteração, quais ações maximizarão as recompensas no futuro. Um exemplo de aplicação dessa técnica é um modelo que joga xadrez, treinado a partir de partidas jogadas com outro jogador.

2.3.1 K-Nearest Neighbours

k-nearest neighbours é um método de aprendizado estatístico que pode ser utilizado tanto para problemas de regressão como de classificação. Ele é um dos métodos não paramétricos mais simples e conhecidos (JAMES et al., 2023). Dado um valor de K e um ponto x_0, o método de regressão *K-nearest neighbours* primeiro identifica as K observações do conjunto treinamento que são mais próximas de x_0, representadas pelo conjunto N_0. Então, f(x_0) é estimado usando a média das observações N_0. Este método pode ser definido pela equação 1.

$$\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i \tag{1}$$

Por ser um método não paramétrico, o *K-nearest neighbours* pode ter sua precisão comprometida quando o número de variáveis categóricas for próximo do número de amostras disponíveis para o treinamento. Este problema é chamado de maldição da dimensionalidade (JAMES et al., 2023).

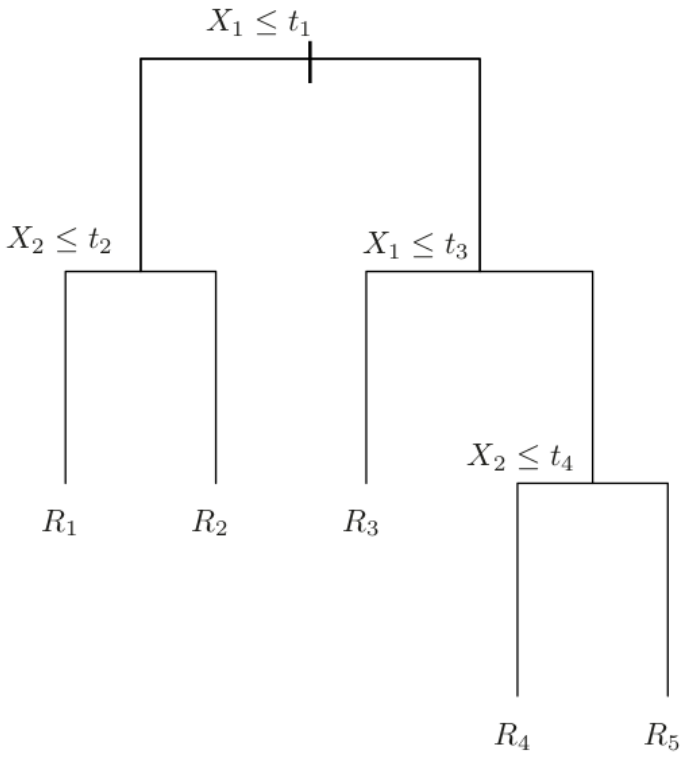
2.3.2 Árvores de Decisão

Uma árvore de decisão é a representação de uma função que associa um vetor de múltiplos valores de entrada a um único valor de saída, chamado de decisão. Uma árvore de decisão gera uma saída realizando uma sequência de testes, começando em um nó raiz e seguindo por diferentes nós até que uma folha. Cada nó representa um teste sobre o valor de uma das varáveis de entrada, e as folhas representam um valor final que é retornado pela função (RUSSELL; NORVIG, 2020).

Árvores de decisão podem ser aplicados tanto a problemas de regressão como de classificação. O processo de treinamento de uma árvore de decisão consiste na criação sucessiva de nós. A cada nó, o conjunto de possíveis valores para uma das entradas é dividido em múltiplas regiões, sendo que cada região corresponde a uma folha ou nó de saída da árvore. No caso de problemas de regressão, previsão do valor de saída para cada região corresponde à média dos valores de saída do conjunto treinamento nesta região. Essas regiões são criadas de modo a minimizar a soma residual dos quadrados das observações do conjunto treinamento contidas na região e a sua média. Este processo é repetido até que um critério de parada seja alcançado, por exemplo, até que nenhuma região tenha mais de cinco observações (JAMES et al., 2023). A Figura 2 mostra uma árvore de decisão com cinco regiões. Para a construção dessa árvore, seleciona-se o preditor X_j e o ponto de corte s tal que a divisão do espaço do preditor nas regiões $\{X|X_j < s\}$ e $\{X|X_j \geq s\}$ leve à maior redução possível na soma residual dos quadrados, e este processo pode ser realizado novamente nas regiões criadas. Na árvore da figura, primeiramente as amostras são divididas utilizando o preditor X_1 e o ponto de corte t_1, criando duas regiões em que $\{X|X_1 \leq t_1\}$ e $\{X|X_1 > t_1\}$. Em seguida, essas duas regiões são divididas novamente,

a da esquerda considerando o preditor X_2 e o ponto de corte t_2 e a da direita o preditor X_1 e o ponto de corte t_3. Por fim, uma das quatro regiões passa por esse processo novamente, sendo divididas utilizando o preditor X_2 e o ponto de corte t_4, criando a árvore com 5 regiões da figura 2.

Figura 2: Exemplo de árvore de decisão



Fonte: (JAMES et al., 2023)

2.3.3 Floresta Aleatória

A Floresta Aleatória é um método de aprendizado estatístico utilizado em problemas de classificação e regressão. Este tipo de método é gerado criando múltiplas árvores de decisão, cada uma utilizando uma parcela do conjunto de dados de treinamento. No entanto, cada vez que um nó é criado durante a construção destas arvores, um conjunto aleatório de m atributos é escolhido para serem candidatos do nó. Tipicamente é escolhido $m \approx \sqrt{p}$ (JAMES et al., 2023). Desse modo, cada divisão da árvore não pode escolher a maioria dos atributos disponíveis, o que diminuí a correlação entre as árvores que compõe a Floresta Aleatória e reduz o sobreajuste (*overfitting*) do modelo.

2.3.4 Redes Neurais Artificiais

De forma geral, redes neurais artificiais são máquinas criadas para modelar o modo no qual o cérebro realiza uma tarefa ou função de interesse. Elas podem ser implementadas por meio de circuitos eletrônicos ou por *software* em um computador. Para alcançar um bom desempenho, as redes neurais empregam uma interconexão de um grande número de simples unidades de processamento, chamadas de "neurônios" (HAYKIN, 2009). A Figura 3 mostra uma rede neural simples do tipo *feed-forward* com uma camada oculta, usada modelar uma resposta quantitativa utilizando p = 4 variáveis de entrada. Redes neu-

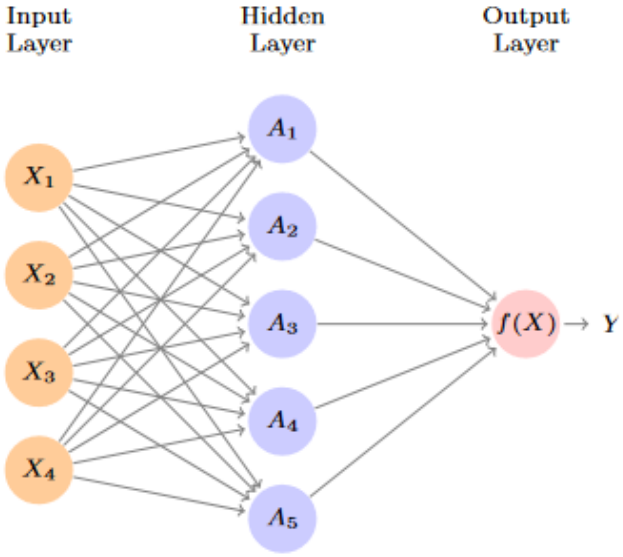
rais *feed-forward*, também chamadas de redes neurais profundas e *multilayer perceptrons*, são os modelos mais tradicionais de aprendizado profundo (GOODFELLOW; BENGIO; COURVILLE, 2016).

Uma rede neural recebe um vetor de entrada de p variáveis X = (X1, X2, ..., Xp) e constrói uma função não linear f(X) para prever a resposta Y. Na rede neural da Figura 3, os 4 neurônios X1,...,X4 compõe a camada de entrada. Cada um deles conecta-se com os K neurônios da camada oculta (K=5 no exemplo da figura), que, por sua vez, servem como entrada para a camada de saída (JAMES et al., 2023). Este modelo de rede neural tem a forma:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) = \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right). \tag{2}$$

Os neurônios da camada oculta são dados por $A_k = h_k(X) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$, em que $g(z)$ é uma função de ativação não linear, definida previamente, e os parâmetros $\beta_0, \ldots, \beta_K$ e $w_{10}, \ldots, w_{kp}$ devem ser estimados a partir dos dados usados para o treinamento do modelo de rede neural. Para respostas quantitativas, tipicamente o erro quadrático é utilizado, de modo que os parâmetros são escolhidos para minimizá-lo.

Figura 3: Rede Neural *Feed-forward*



Fonte: (JAMES et al., 2023)

2.4 Arquitetura TCP/IP

A arquitetura TCP/IP é um conjunto de protocolos de comunicação para redes de computadores. Ela foi desenvolvida nos anos 70, para resolver problemas de integração dos protocolos existentes na época com as novas redes de rádio e satélite. Além disso, o Departamento de Defesa dos Estados Unidos queria que as conexões permanecessem intactas entre as máquinas de origem e destino em caso de falha na operação da infraestrutura intermediaria. Na época, também eram visadas aplicações com requisitos divergentes, como a transferência de arquivos e a transmissão de dados de voz em tempo

real, e para isso era necessária uma arquitetura de redes flexível. Por esses motivos, houve o desenvolvimento do TCP/IP (TANENBAUM; WETHERALL, 2011).

O modelo TCP/IP é importante pois os protocolos definidos por ele são amplamente utilizados nas redes de computadores atualmente. Segundo Kurose e Ross (2013), é possível dividir o modelo de referência TCP/IP em cinco camadas: Aplicação, Transporte, Rede, Enlace e Física. A camada de Aplicação é onde os aplicativos de rede e seus protocolos residem. Ela inclui muitos protocolos, como o HTTP (do inglês *Hypertext Transfer Protocol*), que fornece solicitação e transferência de documentos da Web, SMTP (do inglês *Simple Mail Transport Protocol*), que processa e fornece a transferência de mensagens de e-mail, e FTP (do inglês *File Transfer Protocol*), que fornece a transferência de arquivos entre dois sistemas finais. A camada de transporte da Internet transporta mensagens da camada de Aplicação entre diferentes máquinas. Na Internet existem dois protocolos de transporte, TCP e UDP, qualquer um dos quais pode transportar mensagens da camada de aplicação. A camada de Rede é responsável por mover os pacotes da como datagramas de um *host* para outro. O protocolo IP é o mais importante desta camada. A camada de enlace se preocupa em como enviar pacotes entre computadores conectados diretamente com níveis específicos de confiabilidade. Finalmente, a camada Física tem como função transmitir os datagramas pelo meio físico, na forma de sinais elétricos.

2.5 HTTP

O *Hypertext Transfer Protocol* é o protocolo da camada de aplicação central da *World Wide Web*. Ele é definido nos *Requests for Comments* 1945 e 2616. O HTTP é implementado em dois programas: um programa cliente e um programa servidor. O programa cliente e o programa servidor, executados em sistemas finais diferentes, comunicam-se entre si por meio da troca de mensagens HTTP. O HTTP define a estrutura dessas mensagens e como o cliente e o servidor trocam as mensagens (KUROSE; ROSS, 2013).

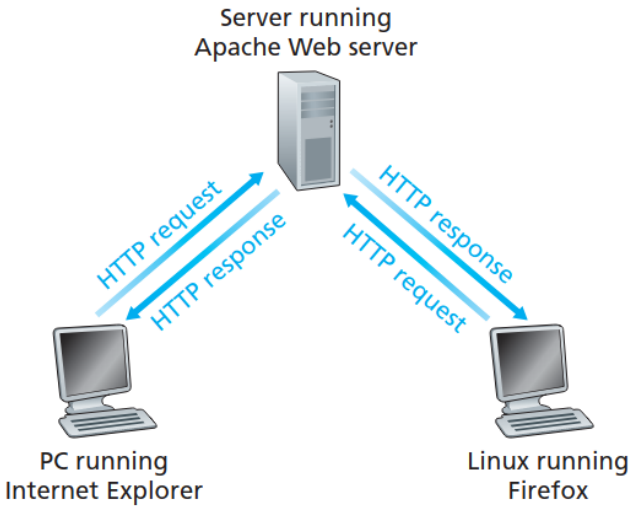
No HTTP, uma página Web (também chamada de documento) é composta de múltiplos objetos. Um objeto é um arquivo acessível por meio de um URL, por exemplo, um documento HTML, uma imagem JPEG ou um vídeo MP4. A maioria das páginas Web é composta por um documento HTML base que pode ter ou não referências a outros objetos. Um URL é um endereço composto por um nome de domínio, que define o servidor, e um caminho de recursos, que define a localização de um documento dentro do servidor.

Navegadores Web, como o *Google Chrome* e o *Firefox*, são aplicações que implementam o lado do cliente do protocolo HTTP, e permitem que os usuários acessem documentos HTML por uma interface gráfica. Servidores Web implementam o lado do servidor do HTTP, hospedam páginas acessíveis por um URL. Pode-se citar como exemplo de servidores Web o *software Nginx*.

O HTTP define como ocorre a comunicação entre clientes e servidores Web. Quando um usuário solicita uma página da Web (por exemplo, clica em um hiperlink), o navegador envia ao servidor uma mensagem de requisição de um recurso. O servidor recebe as requisições e responde uma mensagem de resposta que contém os objetos. O HTTP utiliza o protocolo TCP da camada de transporte para realizar a troca de mensagens entre clientes e servidores. É importante notar que o HTTP é um protocolo sem estado. Um servidor Web não guarda nenhuma informação sobre os clientes, cada par de requisição e resposta entre um cliente um servidor são independentes. A figura 4 mostra a comunicação entre clientes e servidor no HTTP.

Nas aplicações Web, frequentemente a interação entre cliente e servidor envolve o

Figura 4: Comunicação entre clientes e servidor no HTTP



Fonte: (KUROSE; ROSS, 2013)

envio contínuo de requisições e respostas, podendo ocorrer consecutiva, periódica ou intermitentemente. Com o protocolo TCP, é possível enviar cada par requisições e resposta por uma conexão TCP diferente ou enviar múltiplas requisições e respostas pela mesma conexão. No contexto do HTTP, que por padrão adota conexões persistentes, as escolhas impactam as vantagens e desvantagens do design. Examinar esses aspectos é crucial para entender o problema. Embora o HTTP use conexões persistentes em seu modo padrão, Clientes e servidores HTTP podem ser configurados para usar conexões não persistentes (KUROSE; ROSS, 2013).

As especificações do HTTP incluem definições do formato de mensagens usadas no protocolo. Existem dois tipos de mensagens HTTP, mensagens de requisição e mensagens de resposta.

2.5.1 Mensagem de requisição

Uma mensagem de requisição típica tem o seguinte formato:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

A primeira linha de uma mensagem de solicitação HTTP é chamada de linha de requisição; as linhas subsequentes são chamadas de linhas de cabeçalho. A linha de requisição possui três elementos: o método, o URL e a versão HTTP. O método pode assumir vários valores diferentes, como GET, POST, HEAD, PUT e DELETE. O método GET é usado quando o cliente faz uma requisição por um objeto, como no exemplo acima (KUROSE; ROSS, 2013).

As linhas de cabeçalho contém informações sobre a requisição. Elas são compostas por cadeias de caracteres seguidas por dois pontos e valores correspondentes. No exemplo anterior, a linha de cabeçalho Host: www.someschool.edu especifica o onde o objeto está hospedado. Na linha Connection: close, o navegador está informando ao servidor que não quer utilizar conexões persistentes. A linha de cabeçalho User-agent: Mozilla/5.0

especifica o agente do usuário, ou seja, o tipo de navegador que está fazendo a requisição ao servidor. Accept-language: fr indica que o usuário prefere receber uma versão em francês do objeto. No HTTP, existem diversos outros tipos de cabeçalhos além destes citados.

Além da linha de requisição e das linhas de cabeçalho, uma mensagem de requisição também é composta por uma linha em branca após os cabeçalhos, e em seguida um corpo (*body*) contendo dados associados à requisição. Em uma requisição GET, o corpo da mensagem não é usado, porém, em uma requisição POST, o corpo contém dados de formulário HTML enviados pelo usuário quando ele preencheu um formulário na página Web.

Uma requisição GET não envia dados no corpo da mensagem, mas pode incluir dados de formulário no URL. Por exemplo, se um formulário usa o método GET, tem dois campos e as entradas para os dois campos são monkeys e bananas, então a URL terá a estrutura www.somesite.com/animalsearch?monkeys&bananas.

2.5.2 Mensagem de resposta

Uma mensagem de resposta típica do protocolo HTTP é:

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data data ...)
```

A mensagem de resposta tem três elementos: A linha de status, os cabeçalhos e o corpo. O corpo contém o objeto requisitado em si, que pode ser um documento HTML ou uma imagem, por exemplo. A linha de status indica a versão do HTTP que o servidor utiliza, juntamente com um código de status e uma frase associada. Os códigos de status indicam sucesso ou falha da requisição. Alguns valores comuns para eles são:

- 200 OK: A requisição foi bem sucedida e os dados foram enviados na resposta;
- 301 Moved Permanently: O objeto requisitado foi movido permanentemente; o novo URL é especificado em *Location: header* da mensagem de resposta;
- 400 Bad Request: Erro genérico indicando que a requisição não foi entendida pelo servidor;
- 404 Not Found: O objeto requisitado não existe no servidor.
- 505 HTTP Version Not Supported: A versão do protocolo HTTP requisitado não é suportada pelo servidor.

Nos cabeçalhos da mensagem, a linha *Connection: close* informa ao cliente que o servidor fechará a conexão TCP após o envio da mensagem. *Date: Tue, 09 Aug 2011 15:44:04 GMT* indica a data e hora que o servidor enviou a mensagem. *Server: Apache/2.2.3 (CentOS)* indica qual programa gerou a mensagem. *Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT* indica a date e hora que o objeto foi criado ou modificado pela

última vez. *Content-Length: 6821* indica o número de bytes no objeto enviado e, por fim, *Content-Type: text/html* indica o tipo de objeto enviado no corpo da mensagem de resposta (KUROSE; ROSS, 2013).

3 SISTEMA EXISTENTE

3.1 Visão Geral do Sistema

O sistema proposto é uma arquitetura cliente-servidor baseada em painéis que funcionam como clientes. Os painéis são televisores conectados a dispositivos *Raspberrys Pi*, instalados nas estações para exibir informações aos passageiros. Esses painéis fazem requisições a um servidor central localizado na sede da Trensurb. Este servidor armazena informações, processa requisições e gera respostas para os clientes. Neste projeto inicial, o sistema implementado consegue exibir conteúdos de mídia (imagens e vídeos) e uma página com o horário e o nome da estação. O sistema também possui com um programa de administração para que os funcionários da empresa consigam escolher imagens e vídeos para serem exibidos em determinada estação.

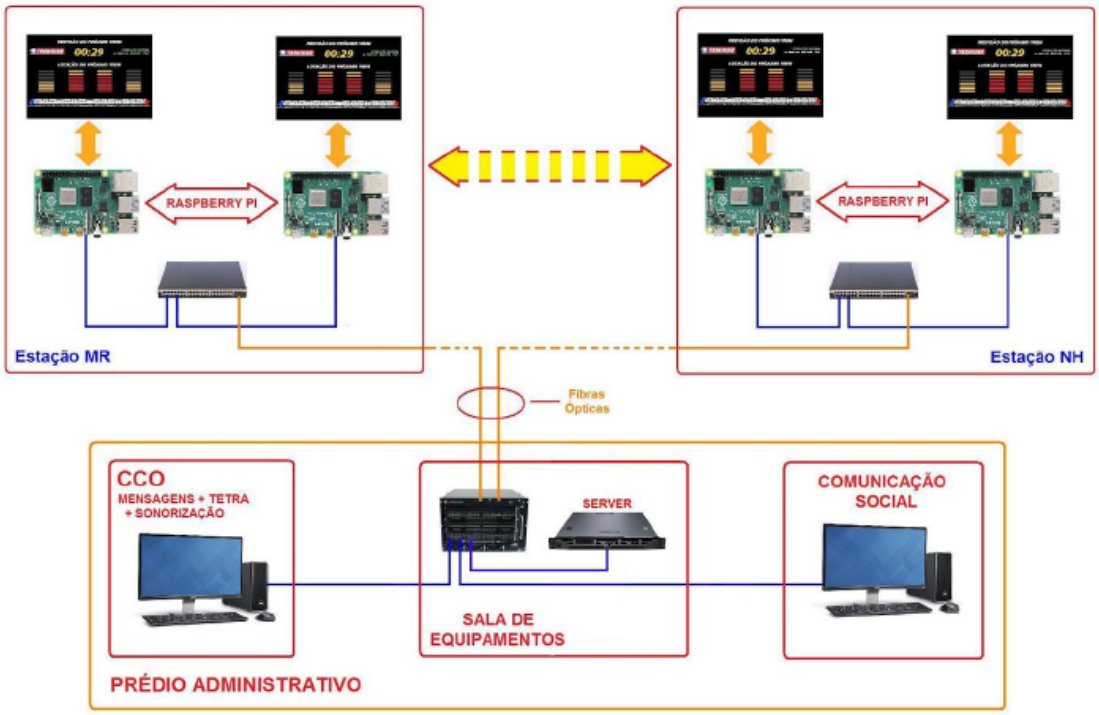
O sistema é dividido em três camadas: uma camada de *back-end* e duas camadas de *front-end*. A camada de *back-end* abrange o controlador, servidor local de backup nas estações, rede interna da Trensurb, servidor geral, seus arquivos e o banco de dados. A primeira camada de *front-end* consiste nos painéis de informações das estações, enquanto a segunda camada de *front-end* é a interface gráfica usada pelo setor de comunicação da Trensurb.

3.2 Módulo Servidor

O módulo servidor está localizado na camada de *back-end* do sistema. Para o seu desenvolvimento foi utilizado o *framework Flask* e o banco de dados MySQL. O servidor foi inicialmente instalado em um computador com o sistema operacional *Windows* 10. O serviço principal deste módulo é o armazenamento e apresentação de informações aos clientes, recebendo e respondendo as suas requisições.

O serviço de apresentação de informações foi implementado utilizando o *framework Flask*, usado para a criação de aplicações Web. Para cada estação foram implementadas dois endereços que os clientes acessam para requisitar os dois tipos conteúdos de conteúdo. O primeiro deles envia aos clientes uma página HTML contendo o nome da estação e o horário atual, e o segundo envia uma página HTML composta por um arquivo de mídia (imagem ou vídeo) em tela cheia. Esses endereços têm o formato «nome da estação>/hor"e «nome da estação>/img_vid». Os dois conteúdos enviam juntamente com o documento HTML um *script*, escrito com a linguagem de programação *JavaScript* que redireciona o cliente à outra página após 8 segundos de exibição, fazendo assim com que as informações exibidas pelo cliente sejam alternadas entre os dois tipos de conteúdo consecutivamente. A Figura 6 mostra a tela de conteúdo de horário para o cliente da Estação Aeroporto, e a Figura 7 mostra um exemplo de imagem com avisos institucionais enviado

Figura 5: Visão geral do sistema proposto



Fonte: (STERNBERG, 2021)

na tela de conteúdos de mídia.

Além disso, o módulo servidor também tem a função de realizar o teste de conexão com os clientes e salvar no banco de dados essa informação. Cada cliente é identificado a partir do IP estático configurado para cada *Raspberry Pi*. Esses endereços IP são armazenados no código do servidor, que periodicamente envia pacotes de teste para os clientes e espera sua resposta para determinar se estão ativos. Há também um segundo teste de conexão realizado pelo servidor, para verificar se houve algum travamento nele mesmo quando conectado à rede. Esse método consiste em um temporizador disparado quando um cliente faz uma requisição ao servidor. Caso um determinado intervalo de tempo transcorra sem que outra requisição seja feita, é considerado que houve um erro de travamento no cliente.

O servidor também realiza a sincronização horária dos clientes com o seu horário, por meio do protocolo NTP e do *software Meinberg*. Esse serviço é necessário, pois o sistema está conectado através de uma rede VLAN isolada.

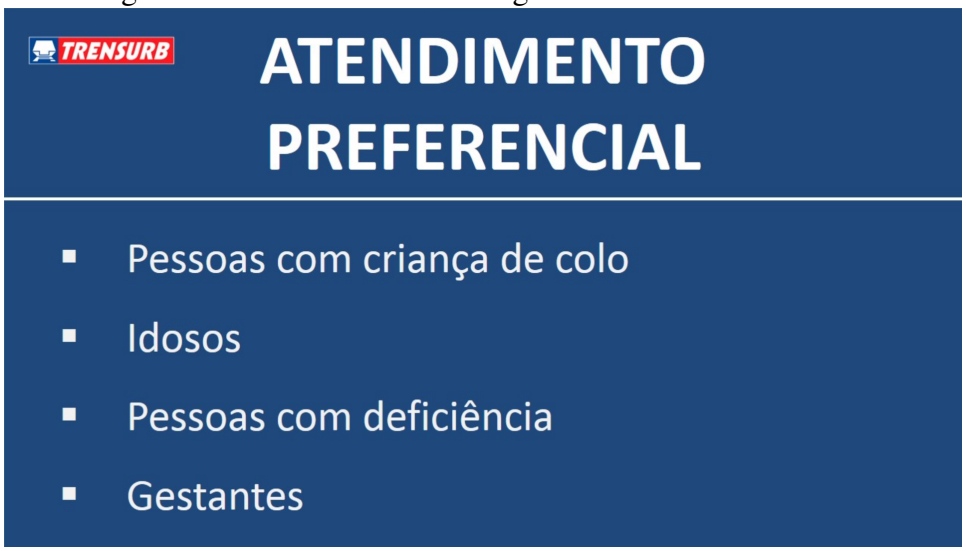
3.3 Módulo Cliente

O *hardware* do módulo cliente é composto de um *Raspberry Pi 3B* conectado à rede VLAN, e um televisor conectado ao *Raspberry Pi* pela saída HDMI. O *Raspberry* atua como um cliente que se conecta ao servidor, enviando requisições via protocolo HTTP, renderiza e exibe as páginas HTML recebidas por meio do navegador *Chromium*. O *Raspberry Pi* foi configurado para abrir o navegador *Chromium* no modo kiosk, de modo que os conteúdos acessados pelo navegador sejam exibidos em tela cheia, e se conectar no endereço do servidor central automaticamente após ser inicializado. A Figura 8 mostra o *Raspberry Pi 3B* utilizado em um dos clientes do sistema.

Figura 6: Tela de horário

Fonte: O autor.

Figura 7: Tela de mídia com imagem de avisos institucionais



Fonte: O autor.

Também foi implementado nele um servidor de backup simples que atua quando não se consegue conexão com o módulo servidor. Este servidor de backup foi criado com o *framework Flask*, e o *Raspberry Pi* foi configurado para rodar este servidor localmente após a sua inicialização. Caso o módulo servidor não responda a uma requisição, o servidor de backup envia uma página HTML com o horário, semelhante à enviada pelo módulo servidor, porém com uma indicação visual de que a conexão com o servidor central foi perdida. Esta página tenta reconectar-se com o módulo servidor periodicamente, testando a conexão com um código em *JavaScript* incorporado à página HTML. Em caso de sucesso, o navegador envia outra requisição ao servidor para continuar a exibir os conteúdos.

O módulo cliente também utiliza o serviço de sincronização de horário, utilizando a ferramenta *timedatect*, um software disponível nos sistemas operacionais Linux. Ele é configurado para fazer periodicamente uma requisição NTP para o servidor e atualizar o relógio interno ao receber a resposta.

Após a implementação dos clientes nas estações, eles apresentaram falhas periódicas relacionadas ao funcionamento do *Raspberry Pi*. Durante essas falhas, ocorria o travamento do seu sistema operacional, impedindo-o de comunicar-se com o servidor e exibir os conteúdos. Observou-se que elas se originavam da corrupção do sistema operacional instalado no cartão SD, que pode ter sido causada por problemas de hardware, interrupção do fornecimento de energia ao equipamento. Essas falhas exigiam a remoção do *Raspberry Pi* e a reinstalação do Sistema Operacional. Além disso, outro problema apresentado pelos clientes foram atrasos e travamentos durante a reprodução de vídeos, por limitações do Hardware do *Raspberry Pi 3B*.

Figura 8: Raspberry Pi

Fonte: (STERNBERG, 2021)

3.4 Módulo de Administração

O módulo de administração é uma aplicação *Desktop*, desenvolvida em *Python* com a biblioteca *Tkinter*, com a função de gerenciar a programação a ser exibida em cada cliente das estações. Este módulo permite que o usuário escolha quais arquivos de mídia serão exibidos em determinado cliente, a partir de uma interface gráfica. Ele também exibe para o usuário as informações de conexão dos clientes das estações, para indicar se há algum

Figura 9: Cliente do prédio administrativo

Fonte: (STERNBERG, 2021)

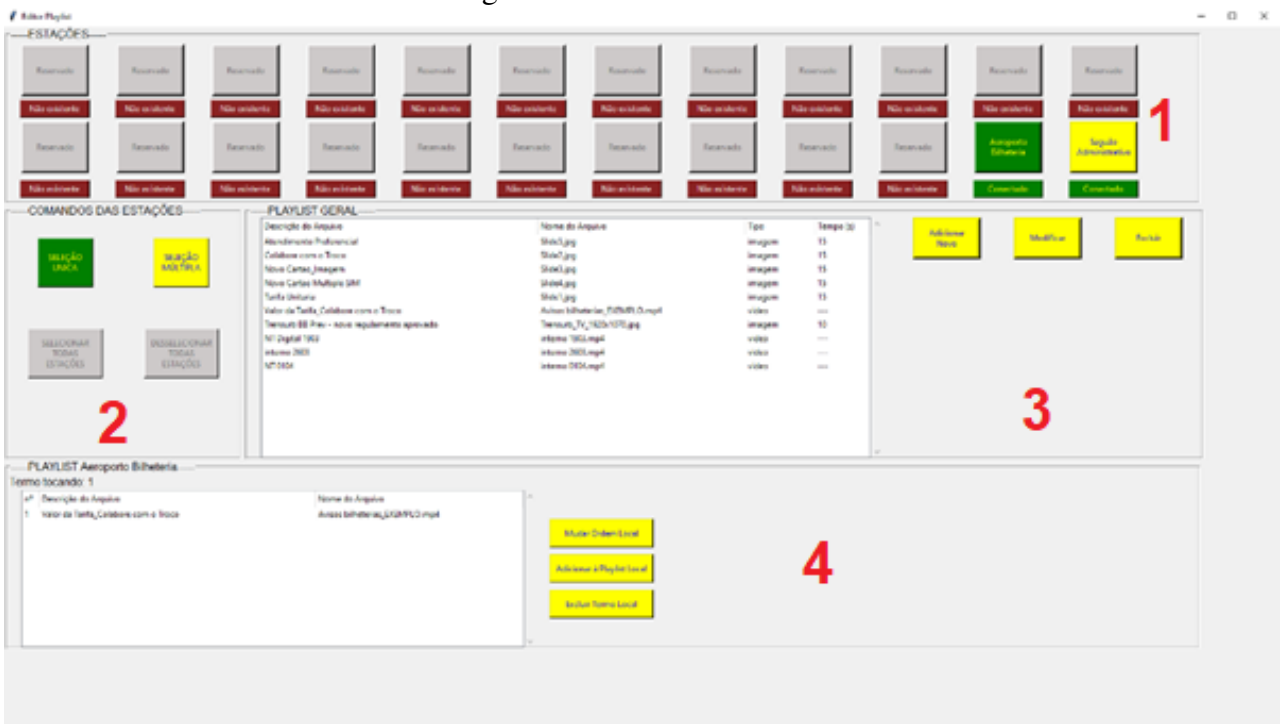
erro em determinado cliente. Este programa deve estar instalado no mesmo computador do servidor, pois se conecta com seus arquivos locais para fazer a administração da ordem de conteúdos a serem exibidos. Por isso, este módulo, juntamente com o servidor, foi instalado na sala do setor GECIN (Gerência de Comunicação Integrada), que coordena a comunicação da empresa com o público externo.

A Figura 10 mostra a tela inicial do programa desenvolvido. O bloco da interface indicado na imagem pelo número 1 exibe diversos blocos menores, sendo cada um deles correspondente a um cliente, com o nome da estação onde ele está localizado. Embaixo de cada bloco há é mostrado o status de conexão do cliente. Ao clicar um dos blocos, ele é selecionado e sua programação é exibida no bloco mais abaixo, indicado pelo número 4.

No bloco 2 há quatro botões: seleção múltipla e seleção única, e em baixo selecionar e desselecionar todas as estações. O bloco indicado pelo número 3 mostra a playlist geral, que é uma lista dos arquivos de mídia presentes no servidor. Neste bloco é possível adicionar, deletar ou modificar um arquivo. Ao clicar em Adicionar Novo, é aberta uma janela do explorador de arquivos para que o usuário escolha um arquivo no computador. Caso seja uma imagem, o usuário também deve escolher a duração de exibição em segundos, caso seja um vídeo, a duração é igual à duração do vídeo. O programa testa se o formato do arquivo é compatível e se a duração é um número válido e então copia o arquivo para um diretório no servidor e atualiza a playlist geral com o novo conteúdo.

O bloco indicado pelo número 4 exibe a playlist da estação selecionada. Nele é possível adicionar um conteúdo da playlist geral, excluir algum conteúdo e modificar a ordem de exibição. Também é mostrado, abaixo do nome da estação, qual conteúdo está sendo exibido na estação.

Figura 10: GUI desenvolvida



Fonte: (STERNBERG, 2021)

4 METODOLOGIA

As melhorias propostas para o aperfeiçoamento do projeto de sistema de informações ao usuário em tempo real são:

- O módulo servidor será reprojetado utilizando o *framework Django*, para usufruir de seus componentes como o mapeamento objeto-relacional (ORM), a interface administrativa e o uso do princípio *Don't repeat Yourself*, que permitirão resolver deficiências de código na implementação antiga e integrar com eficácia as novas funcionalidades;
- Informar o tempo de chegada do trem para a plataforma, por meio de um modelo de previsão e da criação de telas para serem exibidas nas estações;
- A substituição dos *Raspberrys Pi* conectados a televisões por *Smart Tvs* com um aplicativo personalizado instalado;
- Migração do módulo de administração para uma página Web, servida pelo servidor central, que permita o gerenciamento do sistema;
- Criação de um sistema de mensagens de alerta para os painéis, que serão usados pelo CCO.

Os procedimentos para o desenvolvimento dessas melhorias do projeto serão detalhados neste capítulo.

4.1 Atualização do módulo servidor

O servidor original apenas usa o banco de dados para salvar o status de conexão de cada cliente, qual arquivo está sendo exibido, os arquivos programados para cada estação. Visto que a programação de cada estação é acessada a cada requisição dos clientes e também é manipulada pelos usuários do módulo de administração, há um grande volume de acessos a esses dados, somados às novas funcionalidades que exigirão a persistência de dados, o que pode comprometer o desempenho do servidor caso sejam implementados de modo ineficiente. Para esses problemas, decidiu-se utilizar novamente um banco de dados *MySQL* no servidor para guardar e manipular esses dados. Além disso, serão criadas programações para diferentes locais de cada estação, sendo estes: a via 1, a via 2, bilheteria e acesso. Assim, serão ao total 85 programações, quatro para cada estação e mais uma para um cliente localizado no prédio administrativo da sede da empresa.

Uma nova funcionalidade que exigirá que sejam feitas mudanças significativas na arquitetura da lógica de requisição e resposta do servidor, é a adição de novas rotas (padrões de URL programadas para enviar determinado tipo de página Web) na aplicação e

a funcionalidade de permitir que sejam definidos no módulo administrativo quais rotas ou imagens e vídeos serão exibidos em cada estação e local, bem como a ordem de exibição. Somado a isso, o sistema de mensagens de alerta também adicionará uma maior complexidade na implementação da lógica de exibição de conteúdos.

Este capítulo de atualização do módulo servidor versará sobre a implementação na aplicação Web das rotas e lógica de requisição e resposta, do banco de dados, do novo módulo de administração e do novo sistema de mensagens de alerta. A implementação do novo módulo de administração foi colocada neste capítulo porque enquanto no projeto inicial foi feita uma separação entre o módulo servidor e o módulo de administração, neste trabalho o módulo de administração foi agregado ao servidor, sendo implementado como uma página Web.

4.1.1 Rotas e Lógica de Requisição e Resposta

No servidor original a ordem de exibição era sempre a mesma: as rotas de horário e imagem/vídeo eram exibidas alternadamente, e apenas se podia escolher a ordem de arquivos de mídia que a rota de imagem/vídeo exibiria. A rota *hor* era acessada obrigatoriamente, e após 8 segundos o cliente era redirecionado à rota *img_vid*, que acessava o banco de dados e verificava o nome do arquivo de mídia a ser exibido na estação e a sua duração. Assim, o sistema alternava entre a exibição da tela de horário e de uma imagem ou vídeo colocada na programação da estação.

No sistema aprimorado, os conteúdos exibidos em cada cliente será totalmente modificável. Poderá ser escolhidas qual rota será exibida, qual a duração e a ordem de exibição. Inicialmente, as rotas definidas serão:

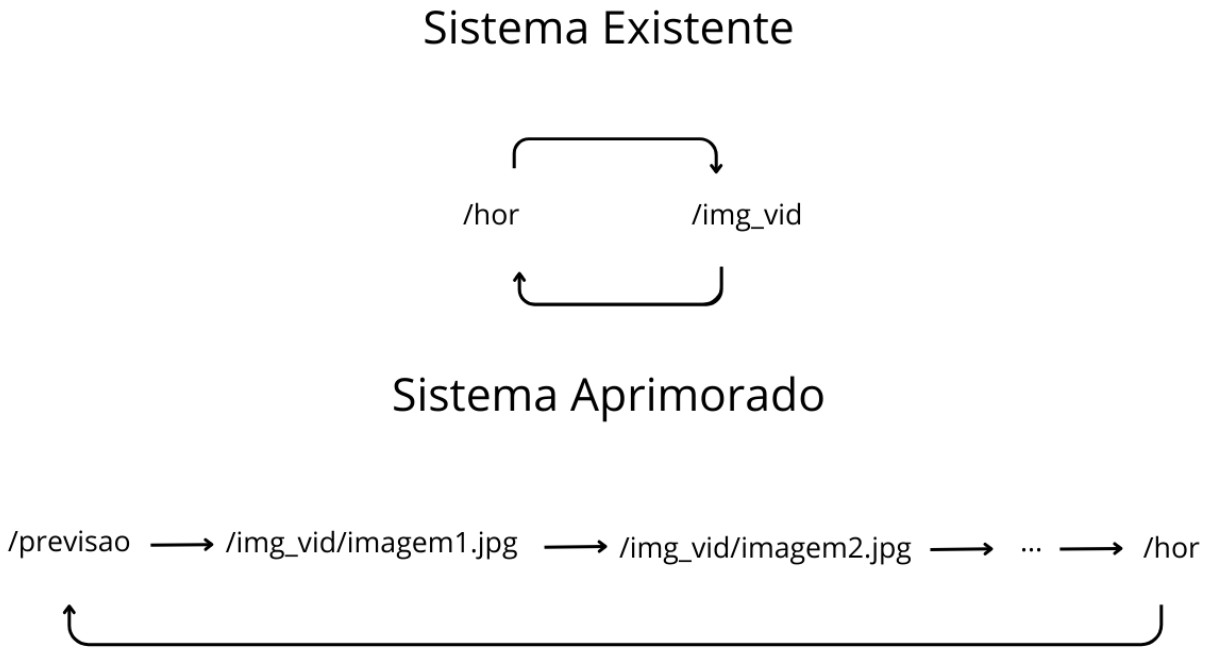
- *hor*: rota que retorna a tela de horário.
- *prediction*: rota que retorna uma tela com a previsão de chegada dos próximos trens para o local correspondente.
- *mapa*: rota que retorna uma tela com uma representação da via com a estação do painel e as duas estações adjacentes, mostrando a localização dos trens neste trecho da via.
- *img_vid/<str:media>*: rota que recebe como argumento o nome do arquivo de mídia e retorna uma tela que exibe o conteúdo do arquivo.

Além dessas rotas de conteúdo, também será definida uma rota chamada *wait*, que será utilizada pelo servidor quando o usuário não definiu nenhuma programação para determinado cliente. Nesse cenário, após o servidor acessar o banco de dados e verificar a ausência de programação, o cliente é redirecionado para este endereço, cuja página é configurada para fazer requisições periódicas ao servidor. Assim, quando um conteúdo é adicionado à programação do cliente, essa página é redirecionada para este determinado conteúdo.

Além disso, no projeto inicial, para cada estação era criado um arquivo separado para as definições de rotas, juntamente com um diretório para os arquivos estáticos. Essa implementação faz com que a base de código do sistema torne-se muito grande, prejudicando a manutenibilidade do código, visto que é prevista a implementação de múltiplos clientes para cada estação. Para corrigir essa deficiência, serão criadas funções genéricas para as cinco rotas citadas anteriormente, que serão usadas por todos os clientes.

Cada programação será definida pelo usuário através da interface gráfica do módulo de administração, e serão salvas no banco de dados. Assim, ao receber uma requisição, cada rota acessará o banco de dados para checar qual é o próximo conteúdo, e enviará juntamente ao seu conteúdo a URL do próximo conteúdo, além da duração de exibição do conteúdo atual. Para que cada rota saiba a estação e o local, devem ser enviados com a requisição parâmetros GET informando esses dados, no formato *rota?estacao=x&local=y* onde x é a sigla da estação e y é o nome do local do cliente.

Figura 11: Lógica de requisição e resposta

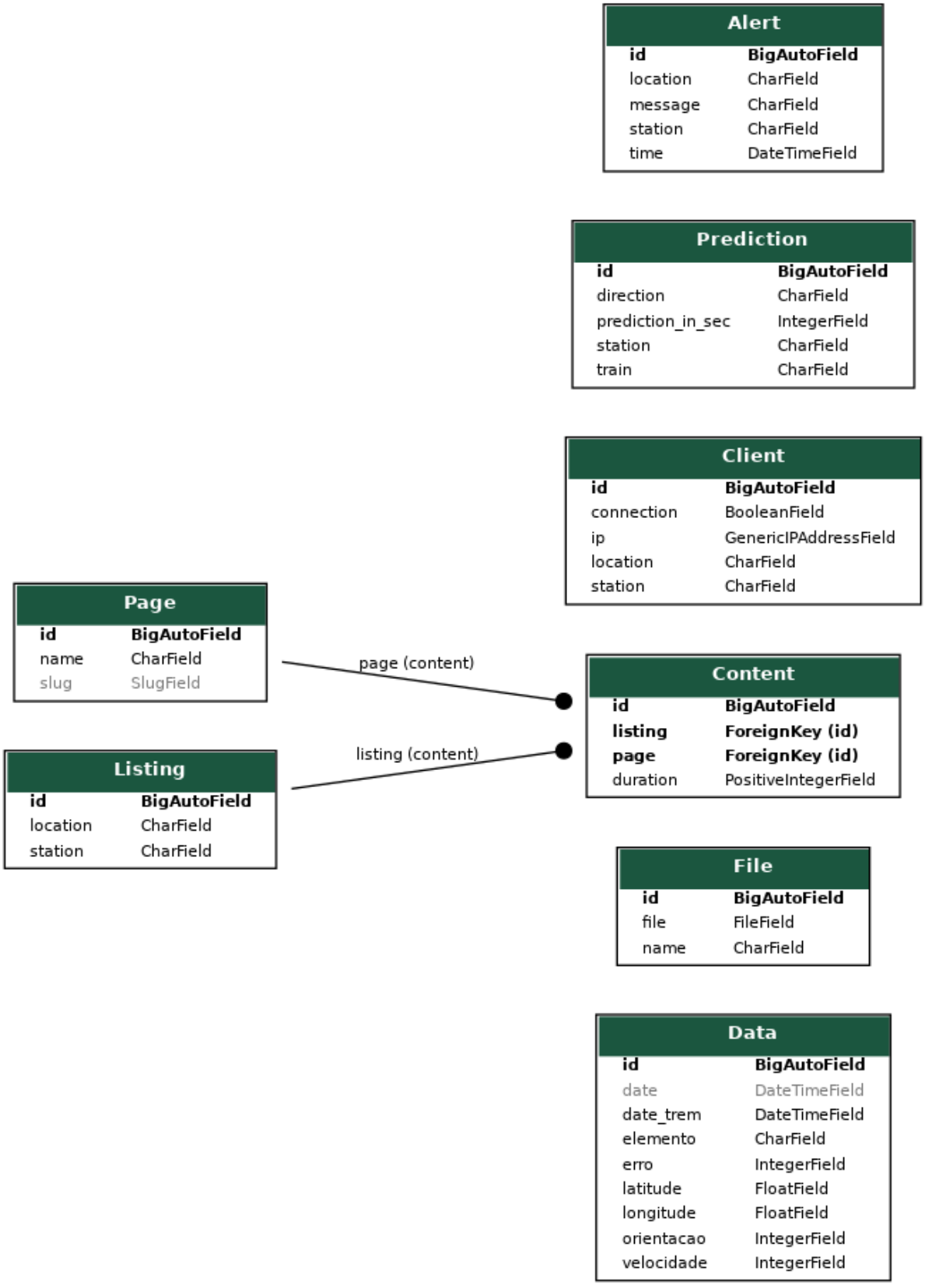


Fonte: O autor

4.1.2 Banco de dados

Para realizar a implementação de um sistema com a lógica e as funcionalidades descritas, os dados utilizados pelos sistemas foram modelados nas entidades mostradas na Figura 12. A entidade *Alert* representa as mensagens de alerta definidas com a interface gráfica pelo CCO. As entidades *Prediction* e *Data* representam, respectivamente, a previsão do tempo de chegada de um trem até uma determinada estação e as informações de GPS de um trem. Ambas as entidades são atualizadas periodicamente pelo servidor ao se conectar com o banco de dados do sistema de GPS dos trens. A entidade *Client* representa um cliente de uma estação, e é usada para manter atualizadas as informações de conexão dos clientes. A entidade *File* é usada para gerenciar os arquivos de mídia carregados para o servidor. Finalmente, a entidade *Content* é usada para ligar uma página, representada pela entidade *Page*, a uma programação de uma estação e local específicos, representada pela entidade *Listing*.

Figura 12: Diagrama do modelo entidade relacionamento



Fonte: O autor

4.1.3 Novo módulo de administração

O módulo de administração do protótipo inicial é um programa *Desktop* criado utilizando a linguagem de programação *Python*, com uma interface gráfica de usuário criada utilizando a biblioteca *Tkinter*. No projeto inicial, este programa foi instalado no mesmo computador que hospeda o módulo servidor, e para gerenciar os arquivos de mídia exibidos ele realiza o acesso aos arquivos locais do servidor. Essa implementação traz deficiências para o projeto, visto que o programa só pode ser acessado em uma máquina, e não pode ser instalado em outros computadores, dado que deve ter acesso aos arquivos locais. Inicialmente, o projeto previa apenas um cliente para cada estação, enquanto o novo sistema proposto neste trabalho prevê múltiplos clientes para diferentes locais de cada estação. Por isso, juntamente com as mudanças propostas na lógica de requisição e resposta do módulo servidor, será necessário modificar a interface e o funcionamento do programa, para comportar os novos requisitos. Assim, o novo módulo de administração deverá:

- Ser acessado em diferentes computadores de diferentes locais da empresa.
- Permitir o envio de imagens e vídeos do computador dos usuários ao módulo servidor.
- Comportar a escolha de conteúdos para múltiplos clientes em uma mesma estação.
- Permitir a escolha de quais páginas de conteúdo serão exibidas em cada cliente, além dos arquivos de mídia.

Para realizar o aprimoramento do módulo de administração segundo esses requisitos, foram levantadas diferentes soluções, como a modificação do programa *Desktop* e o acesso a ele de outros computadores por meio de um software de compartilhamento gráfico de *Desktop*, como o VNC; ou a modificação do programa *Desktop* fazendo-o se comunicar com o servidor por meio do protocolo HTTP e fazer o *upload* de arquivos de mídia.

Diante desse cenário, a solução escolhida foi criar uma aplicação Web para o módulo de administração, integrada ao servidor. Essa solução foi escolhida pois uma aplicação Web pode ser facilmente acessada por meio do navegador por qualquer computador na rede corporativa, sem a necessidade de instalação de nenhum programa adicional e também permitindo realizar a atualização do módulo facilmente, apenas atualizando o servidor. Ademais, a implementação da aplicação Web integrada ao servidor permite a utilização de funcionalidades muito úteis do *framework Django*, como o Mapeamento Objeto-Relacional e a geração de formulários automaticamente através dos modelos de dados. Essas funcionalidades permitirão o desenvolvimento mais rápido e eficiente da aplicação e o cumprimento dos requisitos e novas funcionalidades definidas nas demais seções deste capitulo.

4.1.4 Sistema de Mensagens de Alerta

O sistema de mensagens de alerta tem o objetivo de integrar ao sistema proposto uma forma rápida de comunicação entre o Centro de Controle Operacional e os passageiros, a fim de exibir mensagens de emergência em situações atípicas de operação. Ele deve ter um sistema de login para que apenas funcionários autorizados possam acessar a ferramenta e uma interface que permita ao operador escolher mensagens predefinidas e a estação e

local do cliente no qual essa mensagem será exibida. A duração de exibição das páginas de alerta deverá ser fixa e sua exibição será feita alternadamente com as outras páginas de conteúdo.

Para fazer o sistema de mensagens de alerta, será criada uma ferramenta semelhante ao novo módulo de administração. O sistema será uma página Web, que exigirá autenticação do usuário para ser acessada.

4.2 Novo módulo cliente

O desenvolvimento do novo módulo cliente envolve a integração de uma *Smart TV* com um aplicativo especialmente projetado para se comunicar com o módulo servidor. Isso exigirá a criação de um aplicativo compatível com o sistema operacional *Android TV*. A necessidade dessa inovação é inicialmente justificada pelos problemas enfrentados pelo módulo cliente original, conforme descrito no capítulo anterior. Adicionalmente, a transição é impulsionada pela predominância de *Smart TVs* no mercado atual, especialmente aquelas com telas acima de 50 polegadas, utilizadas em painéis informativos. Utilizar o *hardware* já disponível nestes dispositivos para a implementação do módulo cliente não só otimiza a eficiência do sistema, mas também promove uma solução mais viável economicamente. Essa melhoria simplifica o *hardware* do módulo cliente, passando a ser composto apenas por uma televisão conectada à rede interna da empresa por um cabo *Ethernet*. Desse modo, evita-se as falhas que poderiam ocorrer no *Raspberry Pi*, na fonte de alimentação ou no cartão SD utilizados. Isso é importante visto que é previsto instalar mais de um cliente em todas as 22 estações, de modo que essas falhas recorrentes que foram observadas poderiam tornar o sistema ineficiente ou mesmo inviável devido à necessidade constante de manutenção.

O aplicativo *Android* deve ter um menu onde é feito a definição do endereço de IP do servidor, a porta utilizada pela aplicação Web, a estação e o local da estação em que o cliente está localizado. Após definir essas informações, a aplicação deve redirecionar o servidor para o endereço *http://{IP}:{porta}/wait?estacao={estação}&local={local}*. Esses dados devem ser salvos na memória da televisão, de modo que mesmo se ela for desligada eles sejam salvos. Por fim, ao ser inicializado, o aplicativo deve acessar imediatamente o servidor, e, caso seja necessário mudar algum dos dados definidos, deve ser possível utilizar uma tecla do controle remoto para acessar o menu novamente e modificá-los.

O desenvolvimento desse aplicativo foi realizado utilizando a IDE *Android Studio* versão 2023.1.1, e a linguagem de programação *Kotlin* foi utilizada para a sua codificação. A *Smart TV Philips 55PUG7408* de 55 polegadas foi utilizada para o desenvolvimento e testes da aplicação e para a sua implementação inicial. Após o desenvolvimento da aplicação no *Android Studio*, foi gerado um arquivo APK (do inglês, *Android Application Pack*) para instalação do aplicativo na televisão. Este arquivo será utilizado para instalar futuramente a mesma aplicação nas demais *Smart Tvs* que serão instaladas nas estações.

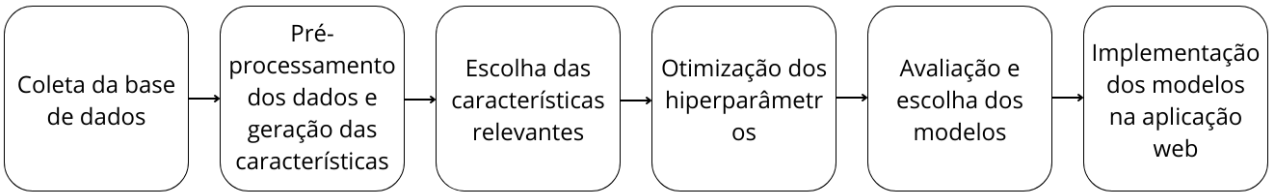
4.3 Criação e Implementação do modelo de previsão para informar o tempo de chegada dos trens

Este trabalho propõe a criação de modelos de previsão do tempo de chegada dos trens nas estações da Trensurb com uma precisão da ordem de segundos, que serão implementados no módulo servidor para exibir essa informação aos passageiros através do sistema projetado. A solução do problema de previsão de chegada dos trens tem algumas pos-

síveis soluções. Nesse trabalho, inicialmente uma solução analítica é proposta, que leva em conta as coordenadas do trem e a velocidade média dos trens na via. Em seguida, serão avaliados métodos de aprendizagem supervisionada com base em bibliografias que contemplam problemas semelhantes.

O procedimento geral desta seção será, primeiramente, a coleta da base de dados, e então o processamento deles para a estruturação de dados de entrada e saída para o modelo. Em seguida será feito o treinamento e a otimização dos hiperparâmetros dos modelos estudados, e então eles serão avaliados em função da sua precisão e também terão seu desempenho comparado com a solução analítica criada. Para realizar o treinamento dos modelos, utilizou-se um computador *Desktop* com o processador AMD Ryzen 5 1600, a placa de vídeo Radeon RX 570 4GB e 16 GB de memória RAM. Por fim, serão escolhidos os melhores modelos e estes serão implementados na aplicação Web da versão final do sistema. A Figura 13 mostra um diagrama do procedimento desta seção.

Figura 13: Fluxograma da metodologia de criação dos modelos



Fonte: O autor

4.3.1 Coleta da Base de Dados

Apesar de haver um sistema que captura as informações de GPS dos trens em tempo real, através do módulo de GPS localizado nos rádios dos trens e da rede TETRA, esses dados não eram armazenados. Assim, há a necessidade de realizar uma coleta de dados para a criação do modelo proposto. Pretende-se obter uma base de dados com as informações de todos os trens obtidas a cada 4 segundos durante um período de uma semana.

No sistema instalado da empresa, cada rádio está configurado para enviar suas informações de GPS a um servidor central a cada 10 segundos, por meio do sistema de rádio TETRA. Este servidor central, à medida que recebe os dados, atualiza um banco de dados que contém as informações de cada trem (identificado por um número de 3 dígitos). Para criar a base de dados, será feito um *script* em *Python* que acessa esse banco de dados a cada 4 segundos e salva todas as informações em um segundo banco de dados, juntamente com a data e a hora de acesso. Esse *script* será executado continuamente por um período de uma semana, a fim de obter uma base de dados que contenha as informações dos trens do sistema metroviário durante este período.

No banco de dados do sistema TETRA, para cada trem do sistema são atualizadas as seguintes informações: O nome do trem, a latitude, a longitude, a velocidade, a orientação, um dado que indica se houve algum erro de medição (1 ou 0), e a data e a hora. Todas essas informações são referentes à última medida realizada pelo módulo de GPS referente à cada trem. No banco de dados criado, serão salvas todas essas informações juntamente com a data e hora de acesso ao banco de dados. Considerando os 40 trens que a empresa possui e o período de aquisição de dados, o banco de dados poderá ter até 6048000 entradas. No entanto, este número deve ser menor, pois nem todos os 40 trens estão operacionais.

4.3.2 Processamento dos dados

Após a criação do banco de dados, deve ser feito o seu processamento a fim de extrair as características de interesse. Este processo será realizado para cada uma das estações, de modo que serão criados 22 *datasets* diferentes. A saída de cada conjunto de características será o tempo de chegada do trem na estação determinada, medido em segundos. As características geradas serão:

- Tipo de trem: trem da série 100 ou série 200;
- Dia da semana;
- Horário do dia: medido em segundos;
- Velocidade do trem em km/h;
- Via: sentido sul ou sentido norte;
- Latitude em graus;
- Longitude em graus;

O banco de dados coletado contém 2 tabelas. A tabela *Trem* contém os atributos de um trem, obtidos em uma determinada leitura. Cada Trem se relaciona a um registro da tabela *Coordenadas*, que contém como atributo o *timestamp* da leitura realizada. A Figura 14 mostra um diagrama do banco de dados coletado. O primeiro passo do processamento é filtrar entradas do banco de dados que não são de interesse, sendo ela entradas com valores repetidos ou entradas que apresentam erros. Para isso, utiliza-se a linguagem SQL para remover linhas da tabela com essas características. Após esta primeira filtragem, serão geradas as saídas para cada amostra, ou seja, o intervalo de tempo entre o instante em que foram obtidos os dados de GPS do trem até o instante de tempo que este mesmo trem chegou à estação. Para isso, implementou-se um algoritmo que, para cada amostra, percorre o banco de dados e determina a qual a amostra com *timestamp* mais próximo em que o trem de mesmo nome chegou na estação. Após determinar a amostra de chegada, o algoritmo calcula o intervalo de tempo em segundos que o trem demorou para chegar na estação e gera as características a partir dos dados da amostra. Este segundo passo é repetido para as 22 estações, e gera um *dataset* para cada uma delas. O último passo é remover amostras que apresentam um tempo de chegada maior que meia hora. Estas amostras não são de interesse para o treinamento dos modelos, visto que o intervalo normal entre trens é de 12 minutos, e o sistema visa estimar o tempo de chegada de no máximo os próximos dois trens para cada estação. Também foram filtrados os dados coletados no período das 23:25 às 5:00, pois durante esse tempo não há circulação de trens para transporte de passageiros.

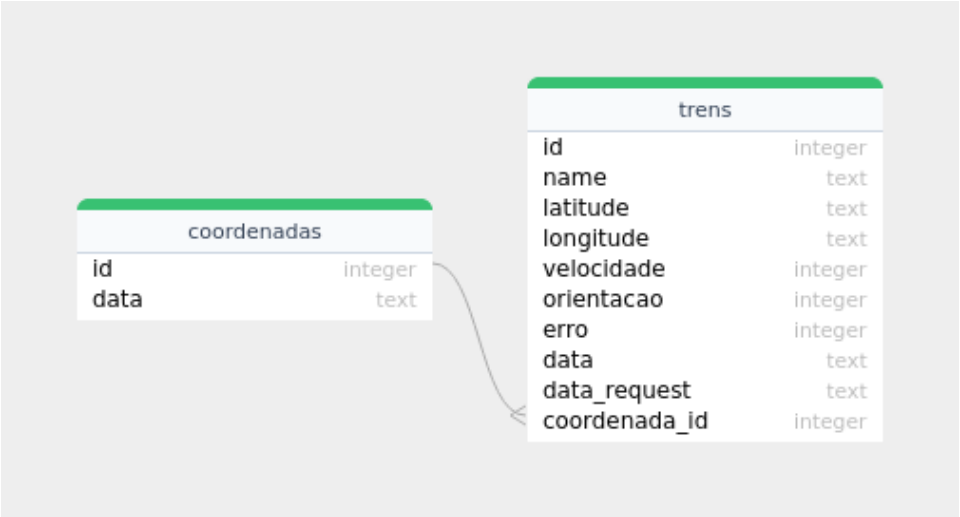
Em média, os *datasets* possuem um total de 285388 amostras. A Tabela 2 mostra o tamanho dos conjuntos de dados gerados para cada estação.

Tabela 2: Número de amostras do conjunto de dados de cada estação

Estação	Número de amostras
AN	258250
AP	277031
CN	334413
ES	320223
FN	247333
FR	286327
FT	330485
IN	261811
LP	263635
MR	221618
MV	326484
NH	221682
NT	327079
PB	304673
RD	234748
RS	313247
SC	257873
SF	281459
SL	328716
SO	327659
SP	224157
UN	329631

Fonte: O autor.

Figura 14: Diagrama de entidade e relacionamento do banco de dados coletado



Fonte: O autor

Depois da geração das amostras com as características especificadas anteriormente, é necessário realizar a transformação dos dados. As variáveis categóricas serão codificadas e as variáveis continuas serão padronizadas. Para as variáveis categóricas nominais, utilizou-se a codificação *one-hot*, com o módulo *sklearn.preprocessing.OneHotEncoder*. Esse método cria uma nova coluna para cada valor diferente encontrado da variável original. Assim, as colunas TrainType, WeekID e Direction foram transformadas em TrainType_2, WeekID_0, WeekID_1, WeekID_2, WeekID_3, WeekID_4, WeekID_5, WeekID_6, Direction_2. Para as variáveis contínuas, utilizou-se o *Standard Scaler*, com o módulo *sklearn.preprocessing.StandardScaler*. Este método dimensiona os dados para terem média igual a zero e desvio padrão igual a 1. A normalização dos dados é importante o lidar com conjuntos de dados contendo diferentes intervalos e unidades de medida, para evitar um desempenho tendencioso do modelo ou dificuldades durante o processo de aprendizagem. Ao fim do processamento, cada amostra do *dataset* possui 13 características, sendo elas:

- TrainType : Tipo do Trem (série 100 ou série 200);
- WeekID_0 : Segunda-feira;
- WeekID_1: Terça-feira;
- WeekID_2: Quarta-feira;
- WeekID_3: Quinta-feira;
- WeekID_4: Sexta-feira;
- WeekID_5: Sábado;
- WeekID_6: Domingo;
- Direction: Sentido de deslocamento (norte ou sul);
- Time: Tempo em segundos desde a meia-noite;
- Lat: Latitude em graus;
- Lon: Longitude em graus;
- Speed: Velocidade do trem em km/h.

Após a obtenção do conjunto de características e saída, esses dados serão divididos em dois conjuntos: treinamento e teste. Decidiu-se usar 70% dos dados para o conjunto treinamento e 30% para o conjunto teste.

4.3.3 Análise das melhores características

Após realizar a divisão dos dados, o conjunto treinamento será utilizado para avaliar quais características são mais relevantes para a criação dos modelos. Este procedimento tem o objetivo de descartar as características que são irrelevantes ou redundantes para a saída. Desse modo, é possível diminuir o tamanho do conjunto de dados de treinamento, e, consequentemente, realizar o treinamento dos modelos mais rapidamente. Para realizar a seleção das características utilizou-se o algoritmo *sequential feature selection*, com o módulo *feature_selection* da biblioteca *scikit-learn*. Este algoritmo adiciona ou remove

iterativamente as características de um *dataset*, avaliando qual delas proporcionou a maior melhoria no desempenho modelo naquela iteração. Para esse algoritmo foi utilizado como estimador uma Floresta Aleatória com profundidade máxima igual a 10.

4.3.4 Otimização dos hiperparâmetros e Análise dos modelos

Para a otimização dos hiperparâmetros será utilizado o algoritmo de busca em grade, que consiste em uma automatização do treinamento utilizando um conjunto de combinações de valores para os hiperparâmetros. Ao fim da busca, a combinação que obteve o melhor desempenho é escolhida. Esse algoritmo foi utilizado com função model_selection.GridSearchCV do módulo *Scikit-learn*. Os modelos estudados serão: rede neural MLP, Floresta Aleatória e KNN (*k-nearest neighbors*). Esses modelos foram escolhidos pois foram algoritmos utilizados em trabalhos relacionados e também por serem algoritmos tipicamente utilizados em problemas de regressão como o abordado neste trabalho. Para cada um dos modelos, serão selecionados múltiplos valores para os parâmetros que os compõe. Visto que foi possível realizar a criação de uma base de dados bastante grande, os valores escolhidos para os hiperparâmetros foram limitados pelo hardware disponível para o treinamento destes modelos. A Tabela 3 mostra os hiperparâmetros dos três modelos estudados e os valores escolhidos para a otimização por meio do algoritmo de busca em grade.

Tabela 3: Valores dos Hiperparâmetros

Algoritmo	Hiperparâmetro	Valores
KNN	Número de vizinhos	4, 5, 6, 10, 20
	Peso	"uniform", "distance"
Floresta Aleatória	Número de árvores	100, 500, 1000, 2000
	Profundidade máxima das árvores	10, 50, 100, None
Redes Neurais MLP	Camadas ocultas	(160, 160),(80, 160), (160, 80), (80, 80)
	Épocas	1000, 2000
	Função de ativação	"relu", "logistic", "tanh"

Fonte: O autor.

Os modelos estudados serão avaliados conforme o erro apresentado, utilizando um conjunto de teste para avaliar a raiz do erro médio quadrático, ou RMSE (do inglês, (*root mean squared error*), A Equação 3 apresenta a RMSE, em que $\hat{f}(x_i)$ é a previsão do modelo para a amostra x_i e y é o valor real. Também será avaliado o tamanho em disco final dos modelos. Além dos modelos de aprendizado de máquina, o conjunto teste também será utilizado para avaliar o desempenho de um algoritmo analítico que estima o tempo de chegada do trem a partir da fórmula do movimento retilíneo uniforme. Esse algoritmo recebe como entrada a latitude e a longitude do trem, e considera velocidade igual à velocidade média de operação dos trens, a qual é 48 km/h. A distância percorrida é calculada a partir das coordenadas do trem até determinada estação, utilizando um arquivo que contém 1000 coordenadas geográficas por onde passa a via da linha férrea da Trensurb.

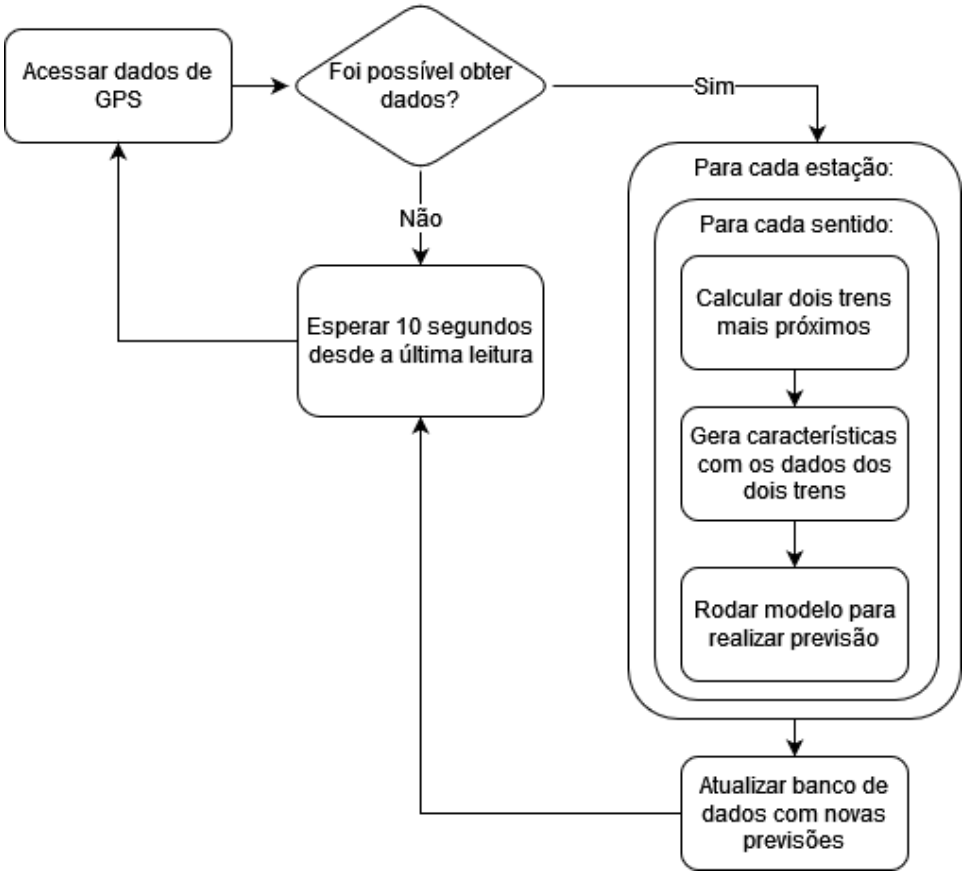
$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2} \tag{3}$$

4.3.5 Implementação dos modelos na aplicação

Após os modelos para cada estação serem escolhidos, os arquivos que contém esses modelos serializados serão utilizados na aplicação Web. Para cada modelo, também foram criados arquivos que realizam o processamento das amostras, serializando os objetos do *StandardScaler* e do *OneHotEncoder*. Esses objetos foram serializados utilizando a biblioteca *Pickle*.

Os modelos treinados foram utilizados no servidor para realizar a previsão de chegada dos próximos trens em cada uma das estações utilizando os dados de GPS em tempo real. Para isso, foi implementada uma *thread* criada na inicialização da aplicação, que realiza operações concorrentemente com a aplicação Web. Esta *thread* realiza periodicamente as operações de acesso das informações de GPS dos trens, cálculo do tempo de chegada dos próximos dois trens para cada estação em ambos os sentidos, e salvar finalmente os resultados no banco de dados.

Figura 15: Fluxograma de previsão



Fonte: O autor

O fluxograma da Figura 8 mostra as operações realizadas pela *thread* para gerar as previsões. Inicialmente o servidor acessa os dados de GPS de todos os trens. Caso a obtenção dos dados não seja bem sucedida, o programa realiza uma nova leitura depois de 10 segundos. Caso sejam obtidos novos dados, inicia-se a fase de previsão. Inicialmente são filtrados dos dados os trens que se localizam fora da via da Trensurb, para que apenas sejam considerados trens em operação. Então, o programa determina os dois trens mais próximos da estação, conforme a sua posição e sentido de deslocamento. Uma vez determinados os dois trens, são geradas características para cada um deles e finalmente

utilizam-se estas características como entrada do modelo de previsão correspondente à estação e, finalmente, a previsão do tempo de chegada para estes dois trens é realizada.

5 RESULTADOS

Neste capitulo, serão primeiramente demonstrados os resultados do desenvolvimento da nova aplicação do módulo servidor. Na seção seguinte, serão apresentados os resultados do desenvolvimento do novo módulo cliente. Na última seção, serão demonstrados os resultados obtidos na coleta de dados e treinamento dos modelos.

5.1 Atualização do Módulo Servidor

No novo sistema, o módulo servidor tornou-se a sua parte mais importante, pois foram adicionadas funcionalidades mais complexas e o módulo de administração passou a fazer parte dele. A versão final da aplicação foi hospedada em um servidor *Windows* da empresa utilizando o módulo Waitress, que implementa um servidor WSGI (do inglês *Web Server Gateway Interface*) para servir aplicações desenvolvidas em *Python*. O servidor tem como endereço de IP 10.0.0.43 e a aplicação foi disponibilizada na porta 5248. Assim, pode-se listar como as suas principais funcionalidades:

- Envio de páginas aos clientes das estações (lógica de requisição e resposta);
- Página de administração;
- Mensagens de alerta;
- Cálculo do tempo de chegada dos trens.

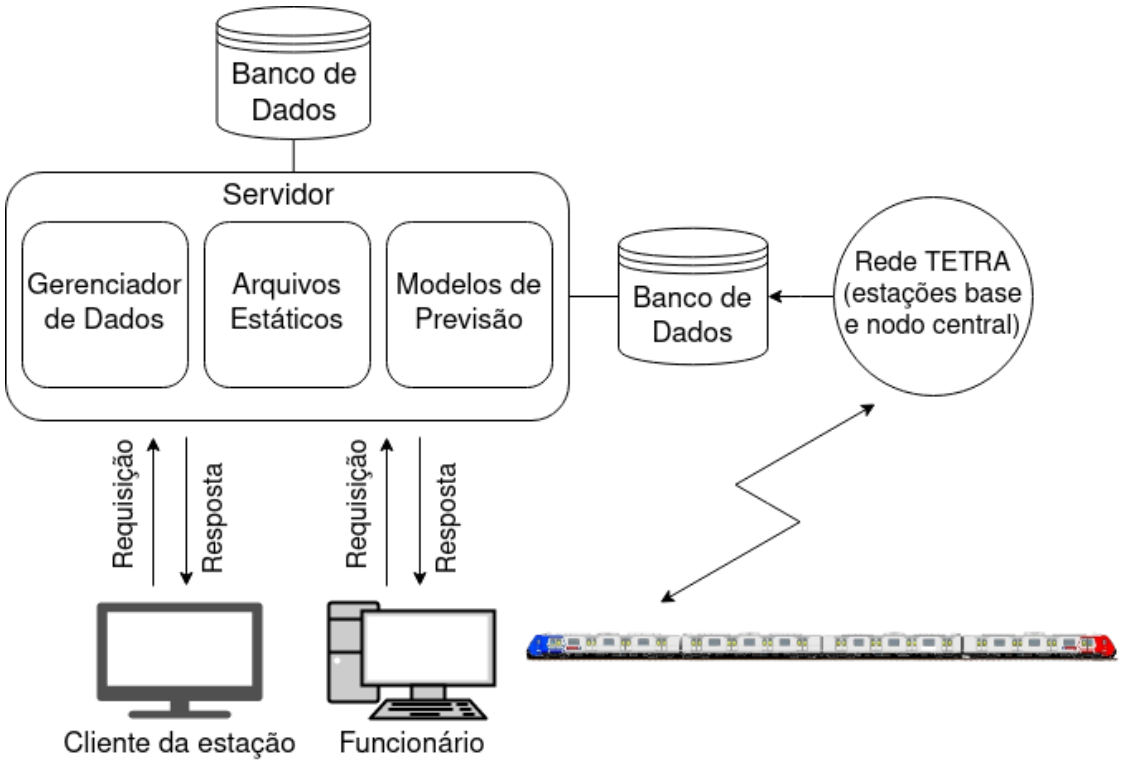
Por se tratar do desenvolvimento de uma aplicação Web, a atualização do módulo servidor usou técnicas de desenvolvimento de software, bibliotecas e *frameworks*. Esta seção aborda os aspectos técnicos do desenvolvimento da aplicação, como as tecnologias, a arquitetura, e a implementação do código e da base de dados, e também demonstra as funcionalidades do sistema final.

A linguagem de programação escolhida para a implementação do novo módulo de administração foi *Python*, por ser a linguagem utilizada na primeira versão do projeto. No entanto, escolheu-se utilizar o *framework Web Django* por possuir mais recursos e funcionalidades que permitem a adição de novas funcionalidades mais rapidamente, porém mantendo a utilização da linguagem *Python*, o que possibilita a reutilização do código desenvolvido na primeira versão. Em particular, os recursos de mapeamento objeto-relacional, interface administrativa, criação e validação de formulários e autenticação de usuários foram importantes para a escolha do *framework*.

Também será necessário utilizar um banco de dados integrado ao sistema, para manter a persistência de dados de forma segura e garantindo a sua integridade. Para isso, foi escolhido o banco de dados *MySQL* por ser um banco de dados de código aberto amplamente

utilizado, bem consolidado e com extensa documentação. Para realizar a comunicação entre a aplicação e o banco de dados, foi utilizado o ORM do *framework Django*. A previsão do tempo de chegada dos trens será feita implementando no servidor os melhores modelos criados, que serão discutidos na seção 5.3 deste capítulo. Como entrada para estes modelos serão usados os dados de GPS dos trens obtidos em tempo real. Esses dados são obtidos de um banco de dados onde o sistema de rádio troncalizado terrestre, implantado na Trensurb pela empresa *Teltronic* em 2020, salva em tempo real os dados recebidos de cada um dos trens em operação. A Figura 16 mostra a arquitetura geral do sistema e como os seus módulos se comunicam.

Figura 16: Arquitetura do novo sistema



Fonte: O autor

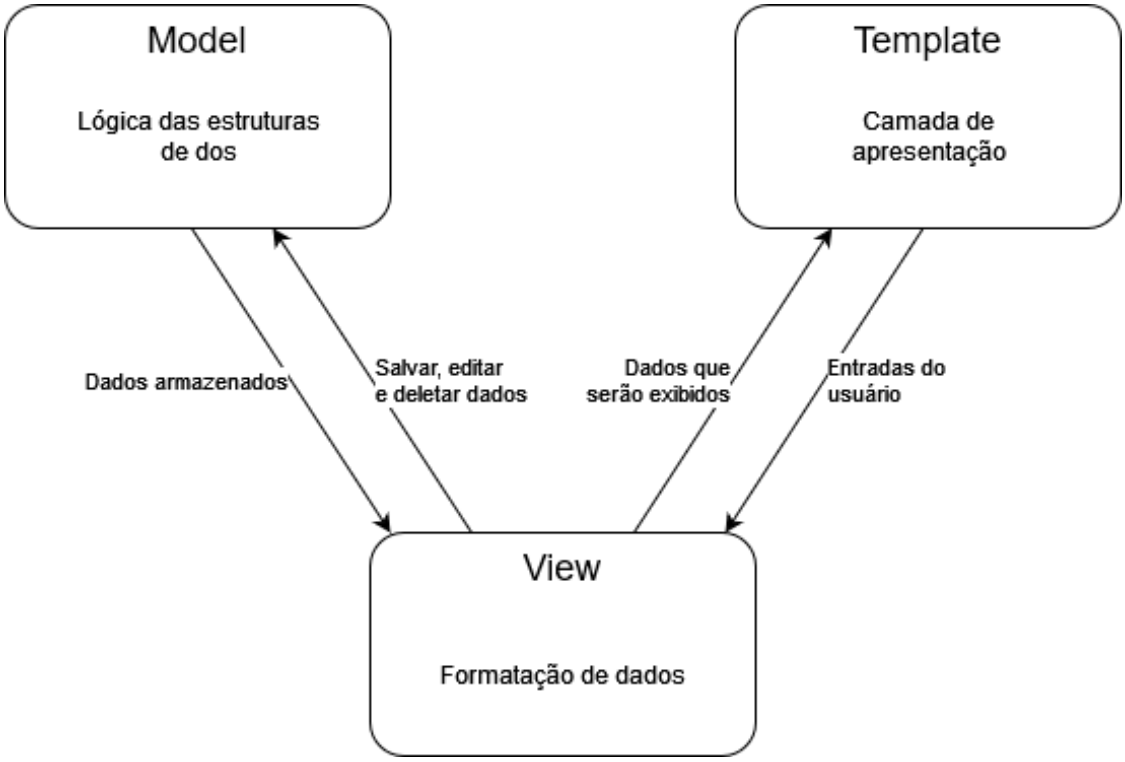
5.1.1 Código

O *framework* Django usa uma arquitetura similar à MVC (Model-View-Controller), chamada MTV (Model-Template-View), por isso o código da aplicação foi organizado seguindo esse paradigma. A Figura 17 mostra um diagrama dessa arquitetura de *software*.

A *View* gerencia a comunicação entre o *Template* e o *Model*. Ele recebe as entradas do usuário, interage com o *Model* para acessar o banco de dados e realiza operações para definir quais informações serão exibidas, e, finalmente, envia ao *Template* os dados corretos. Neste projeto, a *View* corresponde às funções definidas para renderizar as páginas de administração, de mensagens de alerta e de conteúdos exibidas nos clientes, gerenciar as inserções e edição dos conteúdos e das programações e também chamar as funções de login.

O *Model* é responsável pela manipulação dos dados. Ele define a estrutura dos dados, como os campos e as relações entre eles, e também fornece métodos para realizar

Figura 17: Diagrama MTV



Fonte: O autor

operações no banco de dados. A Figura 12 do capítulo anterior mostra os tipos de dados definidos no *Model*. O *framwork Django* permite que sejam feitas consultas e modificações no banco de dados sem precisar implementá-las com código SQL. As consultas são realizadas a partir de uma API chamada ORM (do inglês, *Object Relational Mapping*). Utilizando esta API, são feitas consultas mais otimizadas ao banco de dados e minimiza-se a chance de haver ataques à plataforma, como *SQL-injection*.

O *Template* é a camada de apresentação responsável por definir como os dados da aplicação serão apresentados. Ele contém documentos HTML, com uma pequena quantidade de lógica implementada para realizar a renderização dos dados. Neste projeto, o *Template* realiza a renderização de dois tipos principais de páginas apresentadas: as páginas de gerenciamento dos conteúdos e as páginas de exibição de conteúdo nos clientes das estações. As figuras 18, 19, 20 e 21 mostram, respectivamente, exemplos dos 4 tipos de tela de conteúdo implementadas na nova aplicação.

5.1.2 Rotas e Lógica de Requisição e Resposta

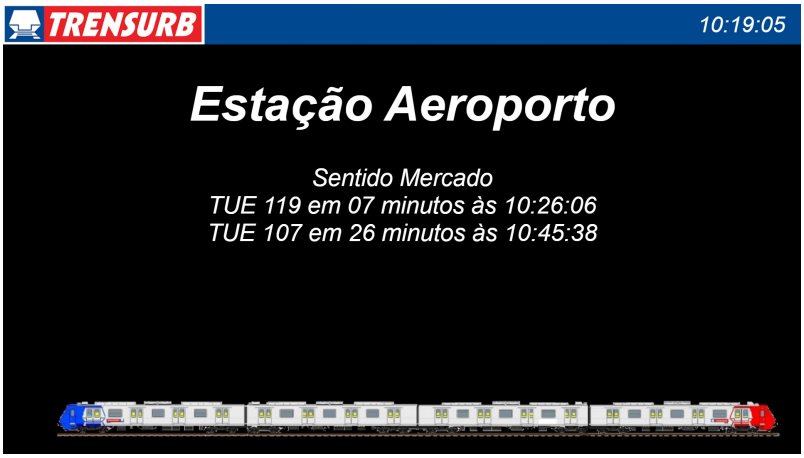
O novo sistema apresenta como requisito permitir a exibição de conteúdos diferentes em determinados clientes em uma mesma estação, a depender de sua localização. Para isso, foram definidas 4 localizações padronizadas: acesso, bilheteria, via 1 e via 2. Ao incluir um cliente no sistema, deve ser definido qual estação e local ele está instalado.

Para implementar essa funcionalidade, definiu-se que a estação e o local do cliente serão comunicados ao servidor utilizando parâmetros GET. Ao requisitar um conteúdo, especificado pelo subdiretório do endereço acessado, deve ser enviados juntos os parâmetros *estacao* e *local*. Quando o servidor recebe a requisição, é verificado se os parâmetros foram enviados. O servidor então acessa o banco de dados para verificar a programação

Figura 18: Tela de horário

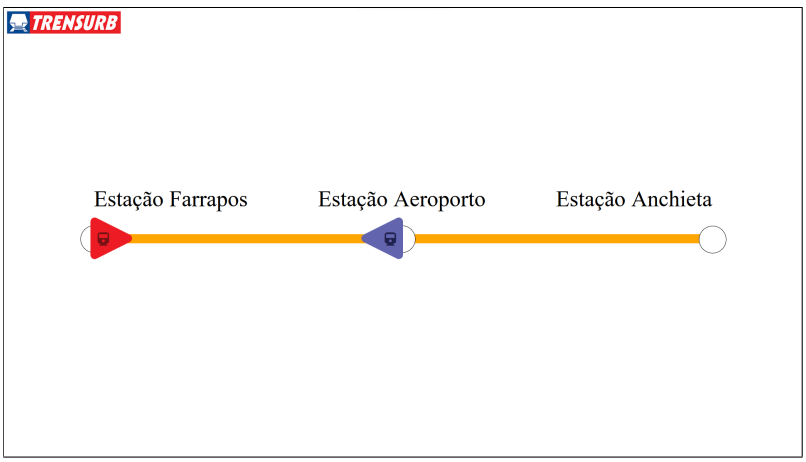
Fonte: O autor

Figura 19: Tela de previsão



Fonte: O autor

Figura 20: Tela com trens no trecho de via



Fonte: O autor

Figura 21: Tela de imagem institucional

Fonte: O autor

correspondente e responde à requisição com o conteúdo adequado ao cliente.

O novo módulo servidor também deve adicionar novas telas de conteúdo e permitir a escolha de quais conteúdos serão exibidos e a sua ordem.

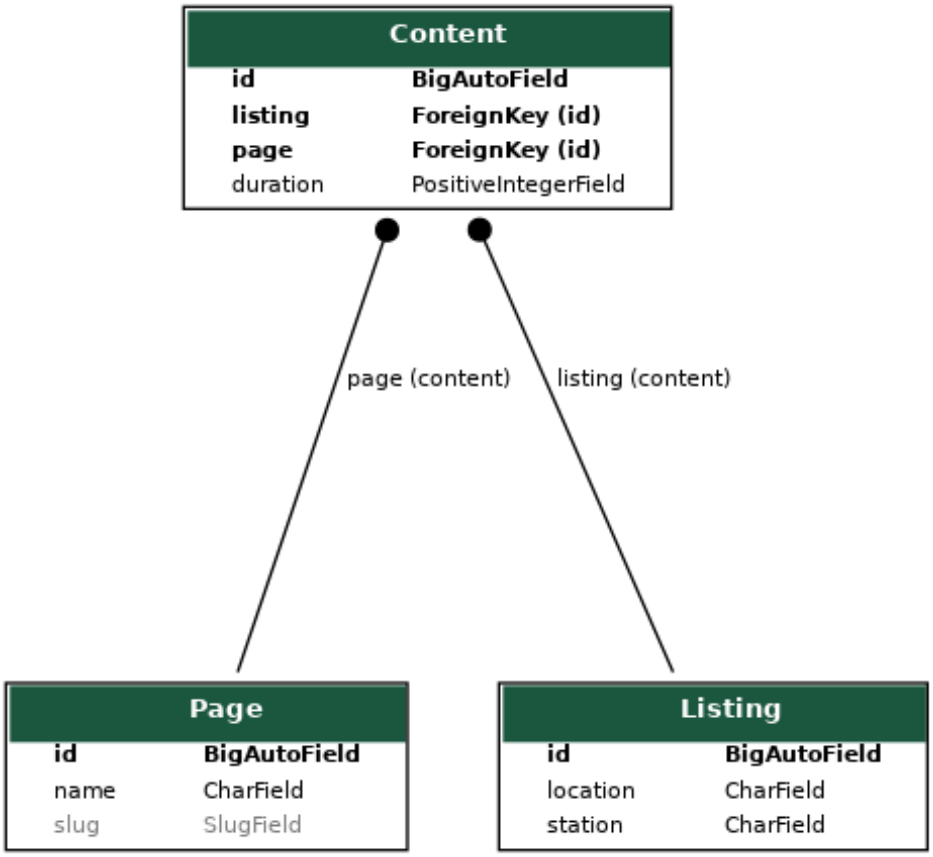
Para implementar essa melhoria no módulo servidor, primeiramente foram criadas entidades no banco de dados para representar a programação de cada estação e local e relacioná-la a múltiplos conteúdos. A entidade *Page* representa uma página de conteúdo, e tem como atributo principal o seu subdiretório. A entidade *Listing* representa a programação de conteúdos de um local de uma estação específica, e possui um atributo que a associa com múltiplas páginas, por meio da entidade *Content*, que relaciona múltiplas *Pages* a múltiplas *Listings* e que também possui um atributo que define a duração de exibição da página de conteúdo na programação. O diagrama da Figura 22 ilustra o relacionamento e os atributo destas três entidades.

Ao receber uma requisição com os parâmetros GET especificados, o servidor acessa o banco de dados para obter o endereço da próxima página de conteúdo na programação e também a duração em segundos de exibição da página solicitada. Então, o servidor responde à requisição do cliente com o conteúdo da página, o qual é composto por um documento HTML com um código escrito na linguagem *JavaScript* incorporado. Este código, que é executado do lado do cliente, possui uma função que redireciona o navegador do cliente para o endereço da próxima página da programação após determinado tempo a partir das informações retiradas do banco de dados. Desse modo, a partir de uma programação definida dinamicamente no banco de dados, cada cliente pode exibir os diferentes tipos de conteúdo (página de horário, página de previsão de chegada, página com mapa da via ou página de mídia) em qualquer ordem definida pelos usuários.

5.1.3 Novo módulo de administração

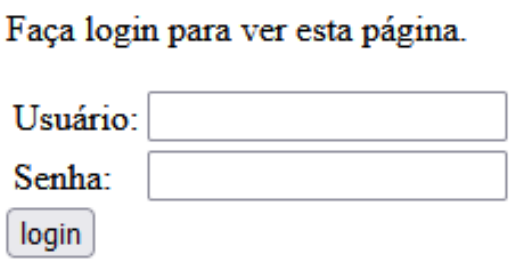
A nova interface de administração, que substitui a aplicação *Desktop* presente no protótipo inicial, foi implementada no servidor. Desse modo, a interface pode ser acessada em qualquer computador conectado à rede interna da empresa. No entanto, para permitir o acesso somente a funcionários autorizados, utilizou-se o sistema de autenticação de usuários para somente permitir o acesso a essa interface a usuários logados. A Figura 23 mostra a tela de login exibida ao usuário que tenta acessar a interface sem estar autenticado.

Figura 22: Relacionamento e atributos



Fonte: O autor

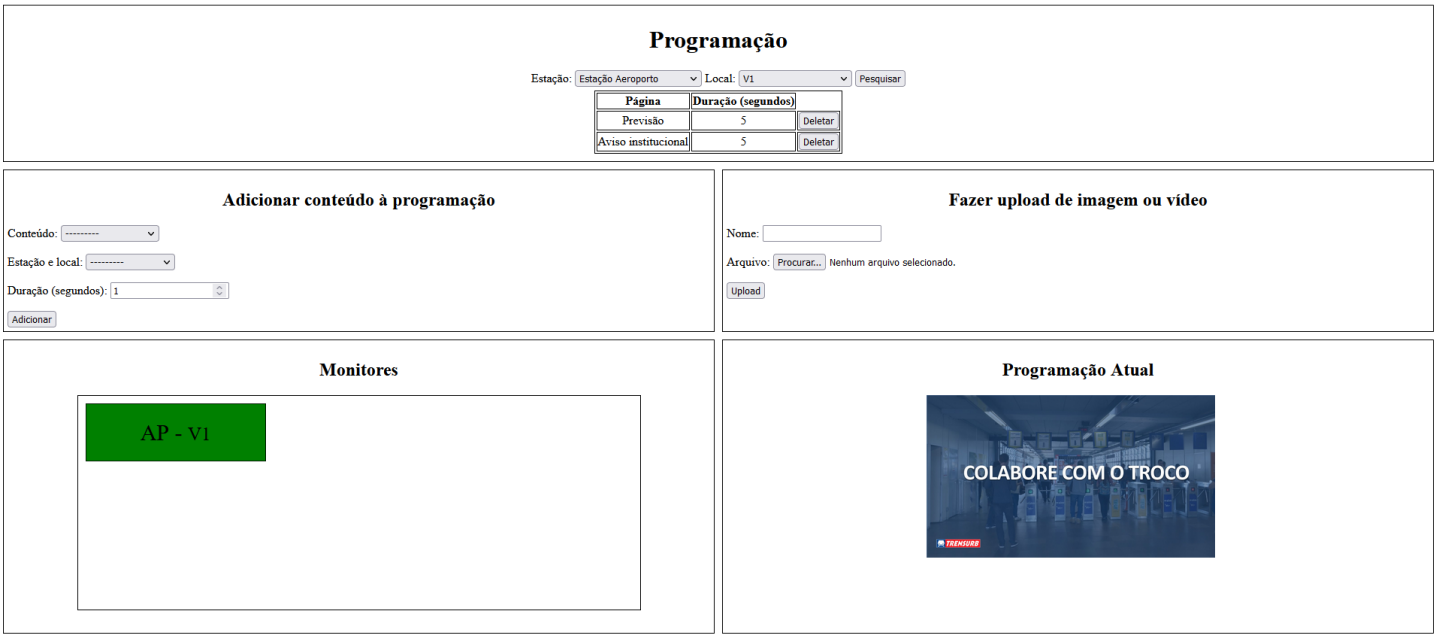
Figura 23: Tela de login



Fonte: O autor

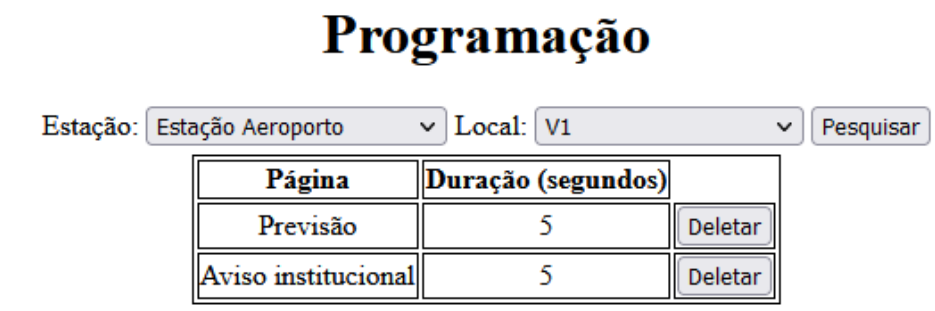
Após o usuário se conectar na aplicação com o seu usuário e senha, a tela da interface de administração é exibida. A interface é dividida em 5 blocos, cada um com uma função diferente para gerenciar o sistema, conforma a Figura 24. No bloco "Programação", o usuário pode conferir a programação de conteúdos de determinado local de uma estação específica. Devem ser escolhidos a estação e o local nos menus de opção, e ao clicar no botão "Pesquisar"a aplicação busca no banco de dados a programação correspondente e a exibe na tabela abaixo. A Figura 25 mostra um exemplo de programação exibida para um painel da estação Aeroporto localizado na via 1. Na coluna à direita da tabela há um botão "Deletar", que possui a função de remover o conteúdo da programação.

Figura 24: Interface administrativa



Fonte: O autor

Figura 25: Bloco de Programação



Fonte: O autor

O bloco “Adicionar conteúdo à programação” tem a função de adicionar conteúdos para serem exibidos nos clientes das estações. O usuário deve escolher o conteúdo a ser adicionado no menu de opções, que contém os conteúdos pré-definidos das telas de

horário e previsão e também as imagens e vídeos enviadas ao servidor pelos usuários. No menu de “Estação e local” o usuário escolhe em qual playlist colocar o conteúdo e em “Duração (segundos)” pode ser escolhida a duração de exibição do conteúdo na programação. A duração mínima é 1 segundo, e a máxima é 1 hora. Caso o usuário tente colocar uma duração fora desta faixa, uma mensagem indicando que a entrada é inválida é exibida. Após clicar em adicionar, o conteúdo é adicionado à programação no banco de dados.

O bloco “Fazer upload de imagem ou vídeo” permite possui um formulário de *upload* de arquivos para o servidor. No campo “nome”, o usuário escolhe um nome para o conteúdo, e no campo arquivo o usuário pode escolher algum arquivo de imagem ou vídeo do seu armazenamento local. O botão “Upload” envia o arquivo ao servidor e faz este novo conteúdo aparecer no menu de opções para ser adicionado a uma programação. Antes de enviar a imagem ao servidor, a aplicação testa se o nome definido já foi utilizado. Neste caso, uma mensagem de erro é exibida ao usuário.

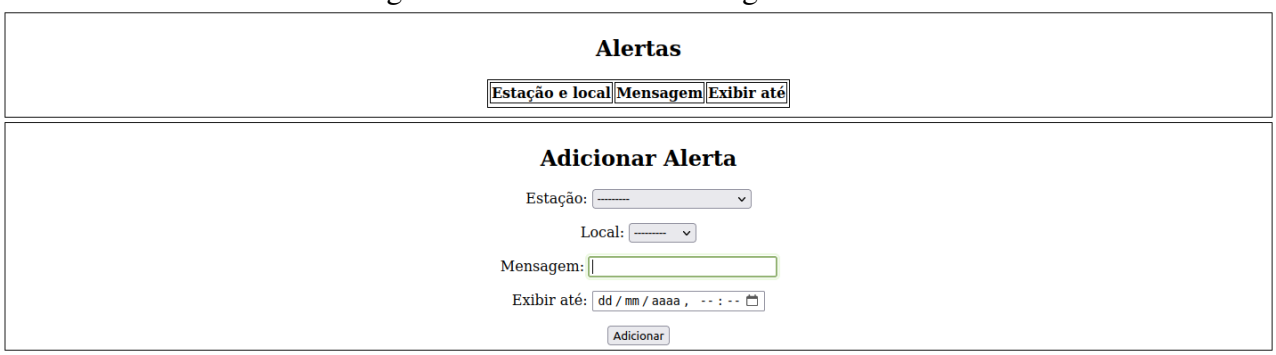
O bloco “Monitores” exibe o status de conexão dos clientes nas estações. Cada cliente deve ser previamente cadastrado no banco de dados, para que o servidor teste a comunicação com este cliente através do protocolo *ping*. A aplicação Web do servidor foi programada para realizar este teste periodicamente, utilizando o campo “IP” de todas as instâncias da entidade *Client* do banco de dados. Caso haja comunicação, a variável “connection” da entidade é atualizada para o valor 1, e o bloco correspondente ao cliente aparece com a cor de fundo verde, caso contrário, a variável “connection” é atualizada para o valor 0, e ele aparece em vermelho.

Por fim, o bloco programação atual exibe um *preview* da programação correspondente ao cliente escolhido no bloco “programação”.

5.1.4 Sistema de Mensagens de Alerta

A interface de gerenciamento do sistema de mensagens de alerta permite ao usuário adicionar e excluir mensagens de alerta para serem exibidas em determinado local. Assim como a interface de administração, para acessar esta página o usuário deve estar logado em uma conta com permissão para realizar este acesso. Caso o usuário não esteja logado, ele é redirecionado para a mesma página mostrada na Figura 23. Com o *framework Django* é possível definir diferentes usuários para o sistema, e também definir permissões para que eles possam acessar ambas as páginas de administração e mensagens de alerta, ou somente a página de administração. A Figura 26 mostra a interface de gerenciamento de mensagens de alerta.

Figura 26: Interface de mensagens de alerta



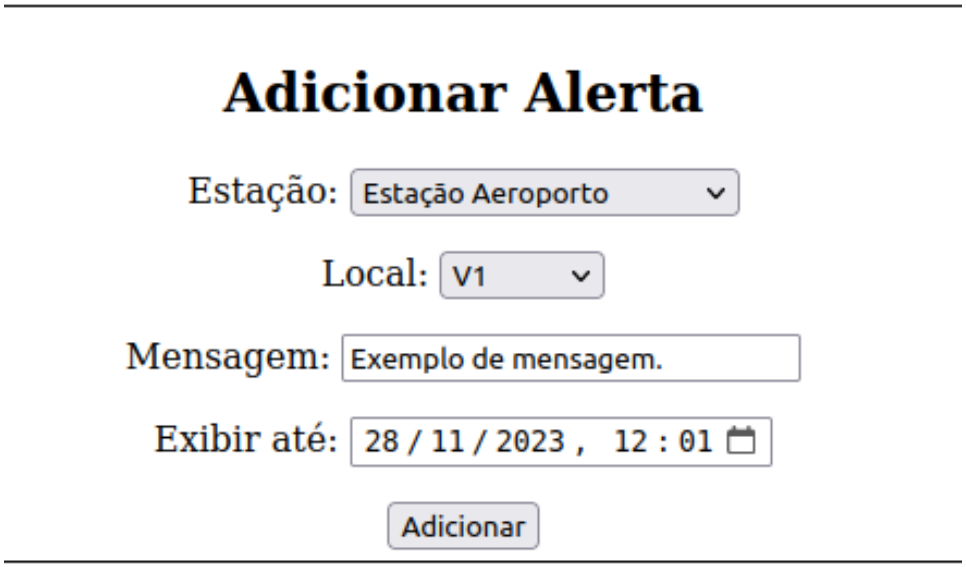
Fonte: O autor

O bloco inferior, com o título de “Adicionar Alerta”, apresenta um formulário para que o usuário solicite ao servidor a adição uma nova mensagem de alerta ao banco de dados, que será exibido no local escolhido. A Figura 27 mostra uma imagem deste bloco. Os campos necessários para a solicitação via formulário são:

- Estação: informa a estação onde a mensagem deve ser exibida. Ela é escolhida por um menu de opções com o nome de todas as estações disponíveis;
- Local: informa o local da estação onde está o cliente em que a mensagem será exibida. Também é escolhido por meio de um menu de opções com os 4 locais permitidos;
- Mensagem: Informa uma mensagem de até 120 caracteres;
- Exibir até: informa uma data e um horário. A mensagem será exibida desde o momento da solicitação até o horário e a data informada.

Ao clicar no botão “adicionar”, a interface verifica se todos os campos foram preenchidos, se a mensagem respeita o limite de 120 caracteres e se a data escolhida é uma data posterior à atual. Caso o usuário tenha preenchido todos os campos com dados válidos, a mensagem é salva no banco de dados, a página é atualizada e o quadro de alertas exibe o novo alerta.

Figura 27: Adicionar alerta



Fonte: O autor

O bloco superior da página, com o título “Alertas”, mostra um quadro com as mensagens de Alerta que foram programadas para serem exibidas. Cada linha da tabela mostra a mensagem, as siglas da estação e do local e a data e horário até os quais a mensagem será exibida. Ao lado de cada linha há um botão para deletar a mensagem antes do horário agendado.

Após a mensagem ser adicionada, ela será exibida alternadamente com as outras páginas de conteúdos adicionadas, e a sua duração de exibição foi fixada em 15 segundos. Ou seja, se houver uma mensagem de alerta programada para determinado painel, após a exibição de cada página de conteúdo programada será exibida a página de alerta por 15

segundos. A Figura 29 mostra a página de mensagens de alerta com um exemplo de mensagem. A próxima seção demonstra exemplos de ordens de programação sem mensagens de alerta e com mensagens de alerta configuradas.

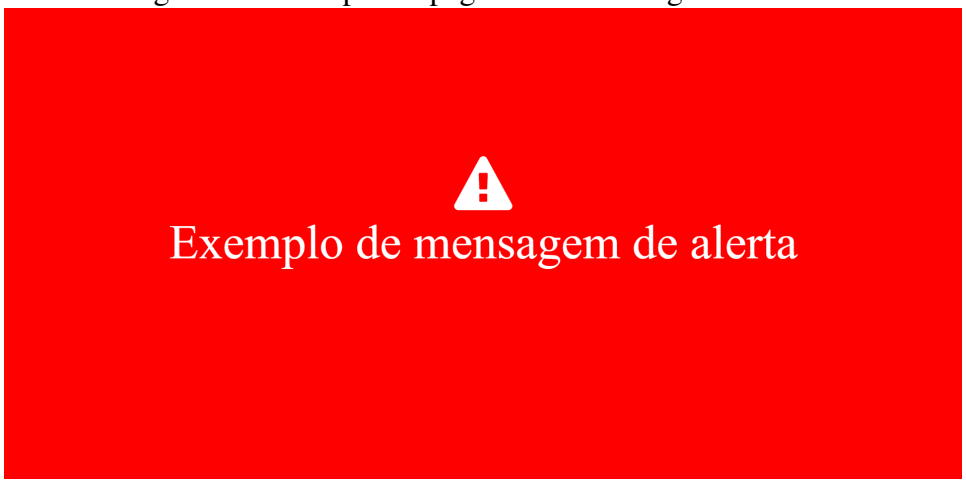
Figura 28: Quadro de mensagens

Alertas

Estação e local	Mensagem	Exibir até	
AP V1	Exemplo de mensagem.	28/11/2023, 12:01:00	Deletar

Fonte: O autor

Figura 29: Exemplo de página com mensagem de alerta

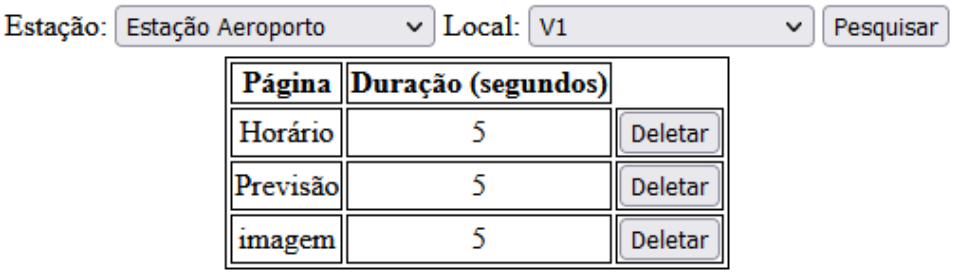


Fonte: O autor

5.1.5 Exemplo de requisição

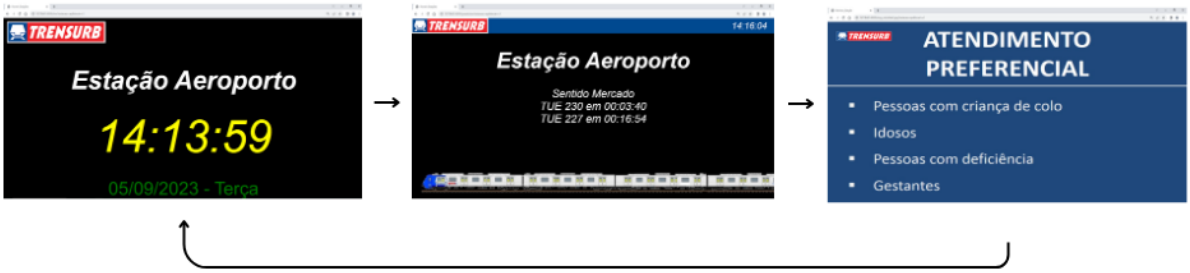
Esta seção detalha um exemplo de sequência de requisições realizado por um cliente localizado na estação Aeroporto, no local da via 1. Primeiramente adicionam-se três objetos *Content* para o banco de dados, o primeiro com a página de horário, o segundo com a página de previsão, e o terceiro com a página de uma imagem institucional, todos com referência para a programação da estação Aeroporto e local via 1. A Figura 30 mostra a tabela de programação para este cliente na página de administração. Então, acessa-se pelo navegador a URL “http://10.0.0.43:5248/hor?estacao=ap&local=v1”. Ao receber a requisição, o servidor busca no banco de dados a programação referente à estação Aeroporto e via 1. Visto que a página de horário está na programação, o servidor responde à requisição com o documento HTML corresponde a essa página, juntamente com um *script* programado para requisitar ao servidor a página de previsão de chegada após 5 segundos. Após exibir o primeiro conteúdo por 5 segundos, o navegador é redirecionado para a página seguinte automaticamente, seguindo a sequência de conteúdos definidos sucessivamente, como mostrado na Figura 31.

Figura 30: Tabela de programação do exemplo



Fonte: O autor

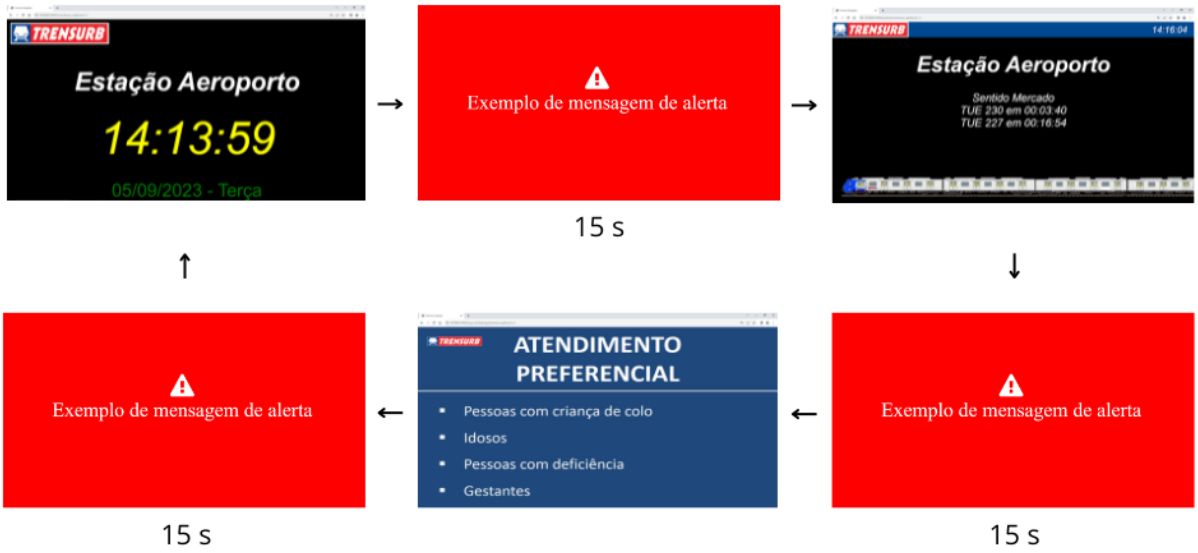
Figura 31: Exemplo de sequência de programação



Fonte: O autor

Em seguida, caso seja adicionada uma mensagem de alerta como a da Figura 28, a exibição das 3 páginas de conteúdo anteriormente definidas será alternada com a exibição da mensagem de alerta por 15 segundos, após cada uma das páginas. Esta ordem de programação é mostrada na Figura 32.

Figura 32: Exemplo de sequência de programação com mensagem de alerta



Fonte: O autor

5.2 Novo módulo cliente

O novo módulo cliente foi desenvolvido no *Android Studio* utilizando a linguagem de programação *Kotlin*. Assim como especificado, após ser aberto, o aplicativo conecta-se com o servidor e exibe a programação de determinada estação e local. A Figura 33 mostra a *Smart TV* utilizado durante o desenvolvimento, com o aplicativo aberto, configurado para a Estação Aeroporto, local Via 1, exibindo a tela de previsão do tempo de chegada dos trens.

O menu para a configuração do painel foi implementado em 3 interfaces. A primeira interface exibe um menu, mostrado na Figura 34, que permite ao usuário definir o endereço de IP do servidor e a porta da aplicação a qual o cliente irá conectar-se. Este menu foi desenvolvido para permitir a fácil reconfiguração dos clientes, caso seja necessário mudar o servidor de hospedagem da aplicação, facilitando assim o acesso ao novo servidor. Após definir o IP e a porta e clicar em salvar, o aplicativo exibe a tela de configuração da estação, mostrada na Figura 35. Esta tela permite que seja escolhida a estação na qual está instalado o cliente. Para selecionar a estação, o usuário deve utilizar as setas do controle remoto para navegar pelos botões contendo as siglas das estações e clicar no botão do meio do controle para escolher a estação. Após a escolha da estação, o usuário é levado à tela de escolha do local, cuja navegação é similar à tela anterior. A Figura 36 mostra a tela de seleção do local. Após a escolha do local, o aplicativo faz uma requisição para o endereço *http://{IP}:{porta}/wait?estacao={estação}&local={local}* e exibe os conteúdos definidos para a estação e local selecionados, como mostrado na Figura 33. Todos os dados escolhidos ficam salvos na memória da televisão, utilizando a API *SharedPreferences*, disponível na *Android Platform*. Desse modo, mesmo que o aplicativo for fechado ou a televisão desligada, ao abrir o aplicativo novamente, os últimos dados definidos estarão salvos e o cliente exibirá automaticamente os conteúdos programados conectando-se com o servidor.

Figura 33: Tela de previsão na *Smart TV*

Fonte: O autor

Figura 34: Menu de seleção de IP e porta

Fonte: O autor

Figura 35: Menu de seleção de estação

Fonte: O autor

Figura 36: Menu de seleção de local

Fonte: O autor

5.3 Modelo de previsão para informar o tempo de chegada dos trens

Nesta seção serão apresentados os resultados obtidos das analises propostas para os modelos de previsão. O desempenho dos modelos treinados a partir das bases de dados apresentadas no capítulo de Metodologia foram comparados entre si e também com o algoritmo analítico criado. Em seguida, também foi avaliado o tamanho em disco dos modelos e discutiu-se a escolha dos modelos a serem implementados no sistema final.

5.3.1 Escolha das melhores características

Para determinar as melhores características, utilizou-se a função *SequentialFeature-Selector* da biblioteca *Scikit-learn* com o algoritmo de Floresta Aleatória como estimador. O número de características a serem selecionadas foi 10. Optou-se por restringir a seleção de características a fim de minimizar o tempo de treinamento dos modelos, considerando os recursos de *hardware* restritos disponíveis. A Tabela 4 mostra as características escolhidas para o *dataset* de cada estação.

Tabela 4: Características escolhidas

Estação	TrainType	WeekID_0	WeekID_1	WeekID_2	WeekID_3	WeekID_4	WeekID_5	WeekID_6	Orientation	Time	Lat	Lon	Speed
AN	x	x				x	x	x	x	x	x	x	x
AP	x		x	x		x	x	x	x	x	x		x
CN	x			x		x	x	x	x	x	x	x	x
ES	x	x	x			x	x	x	x	x	x		x
FN		x			x	x	x	x	x	x	x	x	x
FR	x				x	x	x	x	x	x	x	x	x
FT	x		x			x	x	x	x	x	x	x	x
IN	x	x	x			x	x	x	x	x	x		x
LP	x			x		x	x	x	x	x	x	x	x
MR		x		x	x	x	x	x	x	x	x		x
MV	x	x		x	x	x	x		x	x	x	x	
NH	x	x		x		x	x		x	x	x	x	x
NT	x		x		x	x	x	x	x	x	x		x
PB	x			x		x	x	x	x	x	x	x	x
RD				x	x	x	x	x	x	x	x	x	x
RS	x	x		x	x	x	x	x	x	x	x		
SC	x	x	x			x	x	x	x	x	x		x
SF	x			x	x	x	x	x	x	x	x	x	
SL	x		x		x	x	x	x	x	x	x		x
SO	x		x		x	x	x	x	x	x	x		x
SP	x	x		x	x	x	x	x	x	x	x		
UN	x	x		x		x	x	x	x	x	x		x

Fonte: O autor.

Analisando o conjunto de resultados da seleção das melhores características para todas as estações, nota-se que a orientação, a latitude e o horário do dia foram as características mais importantes para a determinação da saída, pois foram escolhidas em todos os

modelos. O fato da latitude ser escolhida para todas as estações e a latitude em apenas metade delas pode ser explicado pela via da Trensurb seguir uma trajetória longitudinal na maior parte de sua extensão. Já a importância do horário do dia pode ser explicada pelas diferenças de fluxo de passageiros, fatores climáticos e tabela horária de operação ao longo do dia.

Segunda-feira, terça-feira, quarta-feira e quinta-feira foram as características menos relevantes para a saída, enquanto sexta-feira, sábado e domingo foram bastante importantes, não sendo escolhidas em apenas 1,2 e 3 estações, respectivamente. Isso pode ser explicado por sexta-feira haver um fluxo de usuários diferente dos outros dias e sábado e domingo terem tabelas horárias de operação exclusivas para estes dias, conforme as tabelas 5, 6 e 7.

Tabela 5: Tabela horária de partidas da estação Mercado em dias úteis

Faixa horária	Intervalo (minutos)
5h - 6h31	12
6h31 - 6h58	10
6h58 - 7h19	7
7h19 - 16h19	12
16h19 - 16h43	8
16h43 - 18h25	6
18h25 - 18h57	8
18h57 - 19h07	10
19h07 - 19h55	12
19h55 - 20h25	15
20h25 - 23h25	20

Fonte: (TRENSURB - Empresa de Trens Urbanos de Porto Alegre S.A, 2024).

Tabela 6: Tabela horária de partidas da estação Mercado nos sábados

Faixa horária	Intervalo (minutos)
5h - 13h31	12
13h45 - 20h15	15
20h15 - 22h35	20
22h35 - 23h25	25

Fonte: (TRENSURB - Empresa de Trens Urbanos de Porto Alegre S.A, 2024).

Tabela 7: Tabela horária de partidas da estação Mercado nos domingos e feriados

Faixa horária	Intervalo (minutos)
5h - 20h07	20
20h07 - 20h30	23
20h30 - 23h25	25

Fonte: (TRENSURB - Empresa de Trens Urbanos de Porto Alegre S.A, 2024).

5.3.2 Otimização dos hiperparâmetros e treinamento dos modelos

A otimização dos hiperparâmetros foi realizada com o algoritmo de busca em grade, utilizando o método *sklearn.model_selection.GridSearchCV*. Este procedimento foi realizado para o dataset de cada uma das estações, para os três modelos avaliados: *K-nearest neighbours*, Redes Neurais MLP e Floresta Aleatória. A Tabela 8 mostra os resultados obtidos para cada um dos modelos.

Os resultados para os modelos de *K-nearest neighbours* mostram que para o hiperparâmetro peso, o método *distance* foi o melhor valor para todas as estações. Já para o número de vizinhos, a maioria das estações apresentou 4 ou 5 como número ótimo, sendo que apenas as estações Farrapos e Santo afonso apresentaram números maiores.

Os resultados da escolha de hiperparâmetros para redes neurais sugerem uma preferência pela arquitetura de duas camadas ocultas de 160 neurônios cada, escolhida em 10 estações. A taxa de aprendizado adaptativa e um número de iterações apresentaram resultados variados em cada estação, alternando entre os dois valores possíveis.

Por fim, a otimização dos hiperparâmetros para o método de Floresta Aleatória revelou que para metade dos modelos foi escolhido não limitar a profundidade da árvore, enquanto para a outra metade foram escolhidos os valores 50 ou 100, porém em nenhum modelo houve a escolha do valor 10 para este hiperparâmetro. O número de estimadores escolhido foi consistentemente alto, para nenhum modelo foi escolhido o valor 100.

Tabela 8: Hiperparâmetros dos modelos para cada estação

Estação	**K-Nearest Neighbours**		**Rede Neural MLP**			**Floresta Aleatória**	
	Número de vizinhos	peso	Função de ativação	Camadas ocultas	Épocas	Profundidade máxima das árvores	Número de árvores
AN	4	distance	relu	(160, 160)	1000	None	500
AP	4	distance	relu	(80, 160)	1000	None	500
CN	4	distance	tanh	(160, 80)	1000	50	500
ES	4	distance	relu	(160, 80)	1000	50	1000
FN	5	distance	logistic	(160, 80)	2000	None	2000
FR	10	distance	tanh	(160, 160)	2000	None	1000
FT	5	distance	tanh	(160, 80)	1000	None	1000
IN	4	distance	logistic	(160, 160)	2000	50	500
LP	4	distance	logistic	(160, 160)	2000	None	2000
MR	5	distance	tanh	(80, 160)	2000	100	1000
MV	4	distance	tanh	(160, 160)	2000	None	1000
NH	5	distance	tanh	(80, 160)	1000	50	1000
NT	4	distance	tanh	(80, 160)	1000	100	2000
PB	4	distance	relu	(160, 80)	2000	None	500
RD	4	distance	tanh	(80, 160)	1000	None	1000
RS	5	distance	relu	(160, 160)	1000	100	2000
SC	4	distance	relu	(160, 160)	2000	50	500
SF	6	distance	tanh	(160, 160)	2000	50	2000
SL	4	distance	tanh	(80, 160)	1000	100	2000
SO	5	distance	relu	(80, 160)	2000	None	1000
SP	4	distance	tanh	(160, 160)	2000	100	2000
UN	4	distance	relu	(160, 160)	1000	None	2000

Fonte: O autor.

A Tabela 9 mostra a RMSE (raiz do erro quadrático médio, do inglês *root-mean-squared error*) da previsão do tempo de chegada dos trens utilizando o conjunto de dados de teste para os modelos de predição *K-Nearest Neighbors* (KNN), Neural Networks (NN), Random Forest (RF) gerados após o treinamento, e também para o algoritmo analítico. Ao analisar a tabela, observamos diferenças significativas na RMSE entre as estações e também entre os modelos.

O modelo *K-Nearest Neighbors* mostrou um desempenho variável entre as estações, em algumas delas como AN, CN e LP, a RMSE é relativamente baixa, indicando um bom desempenho. Em outras, como SP e SL, o RMSE é muito alto, indicando um desempenho ruim. O modelo de rede neural MLP também apresentou bastante variabilidade entre diferentes estações. Em algumas estações, como ES e PB, sua RMSE é baixa, mas em outras, como SP, o erro é significativamente alta. Já o modelo de Floresta Aleatória apresentou um desempenho mais constante entre diferentes estações.

O algoritmo analítico, em muitos casos, apresenta uma RMSE significativamente maior do que os modelos de aprendizado de máquina. Isso é particularmente notável em estações como AN, AP e MR, onde diferença é aproximadamente o dobro daquela do segundo modelo com menor desempenho. Também pode-se observar que o algoritmo analítico não apresentou o melhor erro em nenhuma estação.

Os modelos de redes neurais mostraram um desempenho geralmente pior em comparação com o *K-Nearest Neighbors*, com RMSEs geralmente maiores. Por exemplo, a estação Matias Velho (MV) teve uma raiz do erro médio quadrático de 72,7197 segundos, quase o dobro do erro para o modelo *K-Nearest Neighbors*. Este padrão se repete em várias estações, como podemos observar nas tabelas.

Por fim, os modelos de Floresta Aleatória apresentaram um desempenho superior aos outros dois modelos em quase todas as métricas e estações. Para a maioria das estações, o seu erro apresentou valores na faixa de 30 a 70 segundos. Isso sugere que o modelo Floresta Aleatória é mais adequado para capturar a complexidade dos padrões contidos no *dataset* criado.

Tabela 9: Raiz do erro médio quadrático dos modelos e algoritmo analítico

Estação	**K-Nearest Neighbours**	**Rede Neural MLP**	**Floresta Aleatória**	**algoritmo analítico**
AN	98.8227	117.9856	61.4197	226.8696
AP	157.655	144.6505	85.4297	258.0964
CN	56.848	69.4666	42.367	142.8626
ES	59.7438	48.3434	36.2215	79.1209
FN	103.5632	96.1955	64.2467	109.8119
FR	144.5436	150.3532	101.4097	207.0816
FT	61.3226	67.2988	45.0892	110.6368
IN	123.392	103.9654	61.3172	113.6521
LP	51.9634	45.3772	34.9443	82.0575
MR	72.2111	81.0402	51.1921	172.1445
MV	38.7942	51.5931	39.3796	93.5118
NH	43.8426	39.1271	31.8769	71.0861
NT	78.6855	80.2045	47.859	116.282
PB	49.4405	44.8908	35.7394	84.2207
RD	90.3937	123.0463	61.8245	166.7389
RS	123.7545	95.8158	64.2788	91.3969
SC	54.4755	50.2076	38.1642	88.3579
SF	112.0971	104.7762	70.925	122.7201
SL	51.0013	51.2316	37.4273	86.4239
SO	120.7595	99.5917	59.8428	85.7544
SP	99.4239	216.5597	93.2506	461.5343
UN	86.4701	100.5529	53.0398	82.2038

Fonte: O autor.

A Tabela 10 mostra o tamanho em KB. A avaliação do tamanho foi feita utilizando o módulo *Pickle*, que tem a função de serializar e desserializar uma estrutura de objeto *Python*. Observa-se que os modelos de Floresta Aleatória apresentaram o maior tamanho, sendo todos estes na ordem de giga bytes, os modelos de *k-nearest neighbours* apresentaram tamanhos na ordem de mega bytes, e os modelos de redes neurais MLP apresentaram os menores tamanhos, na ordem de centenas de kilo bytes. Modelos com tamanhos muito grandes, considerando que 22 deles serão utilizados, podem ser um fator limitante para a sua utilização caso não haja um servidor com armazenamento suficiente. Para a primeira implementação da aplicação em um servidor da empresa, decidiu-se utilizar modelos de *k-nearest neighbours* e rede neural MLP para a maior parte das estações, visto que um servidor com espaço de armazenamento suficiente para utilizar todos os modelos de Floresta

Aleatória não estava prontamente disponível.

Tabela 10: Tamanho dos modelos em KB

Estação	K-Nearest Neighbours	Rede Neural MLP	Floresta Aleatória
AN	4.625	664	1.737.237
AP	4.933	346	1.862.999
CN	5.874	359	2.219.127
ES	5.641	361	4.235.524
FN	5.085	369	6.524.507
FR	5.810	675	3.837.121
FT	4.683	364	4.430.524
IN	4.713	668	1.751.101
LP	4.713	666	7.020.331
MR	4.024	351	2.985.605
MV	5.744	345	4.363.376
NH	4.025	359	2.937.622
NT	5.754	350	8.782.516
PB	5.386	359	2.020.205
RD	4.239	350	3.154.112
RS	5.527	666	8.352.701
SC	4.618	671	1.713.839
SF	5.055	664	7.511.962
SL	5.781	348	8.735.898
SO	5.763	344	4.371.884
SP	4.065	669	6.069.091
UN	5.796	665	8.788.736

Fonte: O autor.

6 CONCLUSÃO

O presente trabalho objetivou a implementação de melhorias e novas funcionalidades no sistema proposto em (STERNBERG, 2021), o qual propunha o sistema para um painel anunciador de informações com atualização em tempo real para estações metroferroviárias do metrô da região metropolitana de Porto Alegre, operado pela empresa Trensurb. Dentre as melhorias e novas funcionalidades, destacam-se as páginas de administração de conteúdos e de mensagens de alerta, as novas telas de conteúdo e os modelos de previsão do tempo de chegada dos trens nas estações, criados para cada uma das 22 estações.

As páginas de administração de conteúdos e de mensagens de alerta e as novas telas de conteúdo foram implementadas a partir de uma refatoração da aplicação Web do módulo servidor do projeto original. A nova aplicação foi construída com o *framework Django* e o banco de dados *MySQL*. O novo sistema permite que os conteúdos exibidos nos painéis da estação sejam administrados mais eficientemente pelos setores da empresa, de modo a melhorar a comunicação com os passageiros do metrô. Além disso, a partir da implementação das principais funcionalidades do sistema na aplicação Web, foi obter um sistema de baixo custo e facilmente expansível.

Foi possível substituir os *Raspberrys Pi* pela *Smart TV* com o aplicativo criado. Essa mudança tornou o módulo servidor menos propenso a falhas, mais flexível se comparado ao anterior, por exigir o uso de um único aplicativo que pode ser configurado e instalado nas *Smart TVs* de qualquer estação, e mais barato por remover a necessidade de equipamentos como o *Raspberry Pi*, fonte, cabo HDMI e cartão SD. Embora inicialmente espera-se implementar clientes do sistema apenas na Estação Aeroporto, no futuro serão adicionados painéis em todas as estações.

Em relação aos modelos de previsão do tempo de chegada dos trens nas estações, o primeiro passo foi a criação de um conjunto de dados para cada estação, contendo dados de localização, velocidade e orientação dos trens durante o período de uma semana, juntamente com uma variável de saída correspondente ao intervalo em segundos entre a obtenção da amostra e a chegada do trem à estação. os conjuntos de dados de cada estação foram utilizados para construir modelos com as técnicas de aprendizado de máquina *k-nearest neighbours*, redes neurais MLP e Floresta Aleatória. Para cada técnica avaliada, foram consideradas diferentes combinações de hiperparâmetros para encontrar o melhor conjunto. Para avaliar os modelos selecionados, mediu-se a raiz do erro médio quadrático apresentado por cada modelo utilizando o conjunto teste. Os modelos também foram comparados com o erro apresentado por um algoritmo analítico testado também com o conjunto teste. Posteriormente, também foram avaliados os tamanhos dos modelos gerados.

A análise quantitativa dos modelos de previsão mostrou que a Floresta Aleatória apresentou os melhores resultados, considerando a raiz do erro médio quadrático, para todas

as estações, exceto a estação Matias velho, na qual o modelo *k-nearest neighbours* apresentou o melhor desempenho. Dentre os modelos que utilizam as técnicas de aprendizado de máquinas estudadas, em média o *k-nearest neighbours* apresentou o segundo melhor desempenho e a rede neural MLP o pior desempenho. De forma geral, o algoritmo analítico apresentou o maior erro dentre os modelos, excetuando-se quatro estações nas quais ele apresentou o segundo melhor desempenho.

Após o treinamento e a análise dos modelos, os que apresentaram melhor desempenho foram integrados à aplicação Web. A partir dos dados de GPS dos trens, coletados em tempo real pelos rádios digitais do sistema TETRA, a aplicação consegue determinar os trens mais próximos de cada estação e prever a partir dos modelos criados o tempo de chegada destes trens às estações e exibir aos usuários esta informação em tempo real. O sistema desenvolvido foi capaz de implementar novas funcionalidade e integrá-las ao projeto original, melhorando-o para atender de forma mais eficiente às necessidades da empresa.

Embora o projeto apresentado neste trabalho tenha conseguido aperfeiçoar o sistema inicial conforme os requisitos levantados, ainda há espaço para a realização de novos estudos e a implementação de mais funcionalidades. Neste sentido, é possível destacar algumas melhorias que podem ser realizadas em trabalhos futuros:

- A implementação de uma tela informando a quantidade de passageiros em cada vagão do próximo trem a chegar na estação.

- Estudar novos métodos de previsão do tempo de chegada dos trens, como o uso de redes neurais *Long Short-Term Memory*, que são um tipo de Rede Neural Recorrente bastante utilizado em problemas que trabalham com séries temporais, e a utilização de um conjunto de dados com mais características, como, por exemplo, dados meteorológicos.

- Adicionar mais funções ao aplicativo para as *Smart TVs* do módulo cliente, como um *backup* de informações em caso de falha de comunicação com o módulo servidor.

REFERÊNCIAS

ANDROID API reference | Android Developers. Disponível em: <https://developer.android.com/reference/>. Acesso em 14 de janeiro de 2024.

BUIJSE, B. J.; RESHADAT, V.; ENZING, O. W. A Deep Learning-Based Approach for Train Arrival Time Prediction. In: INTELLIGENT DATA ENGINEERING AND AUTOMATED LEARNING – IDEAL 2021, 2021, Cham. **Anais...** Springer International Publishing, 2021. p.213–222.

CAULFIELD, B.; O'MAHONY, M. An Examination of the Public Transport Information Requirements of Users. **IEEE Transactions on Intelligent Transportation Systems**, [S.l.], v.8, n.1, p.21–30, 2007.

DJANGO — djangoproject.com. Disponível em: <https://www.djangoproject.com/>. Acesso em 25 de janeiro de 2024.

DUNLOP, J.; GIRMA, D.; IRVINE, J. **Digital Mobile Communications and the TETRA System**. [S.l.]: Wiley, 1999.

GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, MA, USA: MIT Press, 2016.

HAYKIN, S. S. **Neural networks and learning machines**. 3.ed. Upper Saddle River, NJ: Pearson Education, 2009.

JAMES, G. et al. **An Introduction to Statistical Learning with Applications in Python**. Cham: Springer, 2023. (Springer Texts in Statistics).

KOSOLSOMBAT, S.; LIMPRASERT, W. Arrival Time Prediction and Train Tracking Analysis. In: TRENDS IN ARTIFICIAL INTELLIGENCE: PRICAI 2016 WORKSHOPS, 2017, Cham. **Anais...** Springer International Publishing, 2017. p.170–177.

KUROSE, J.; ROSS, K. **Computer Networking**: a top-down approach. [S.l.]: Pearson, 2013. (Always learning).

LOBO, C. **CPTM começa a implantar painéis interativos de LED em duas de suas estações**. Disponível em: <https://www.metrocptm.com.br/cptm-comeca-a-implantar-de-paineis-interativos-de-led-em-duas-de-suas-estacoes/>. Acesso em 25 de janeiro de 2024.

PONGNUMKUL, S. et al. Improving arrival time prediction of Thailand's passenger trains using historical travel times. In: INTERNATIONAL JOINT CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING (JCSSE), 2014., 2014. **Anais...** [S.l.: s.n.], 2014. p.307–312.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence**: a modern approach. [S.l.]: Pearson, 2020. (Pearson series in artificial intelligence).

SCIKIT-LEARN: machine learning in python - scikit-learn 1.4.0 documentation. Disponível em: <https://scikit-learn.org/stable/>. Acesso em 25 de janeiro de 2024.

SKOGLUND, T. Investigating the impacts of ICT-mediated services — The case of public transport traveller information. , [S.l.], 2012.

STERNBERG, L. S. Desenvolvimento de painel anunciador de informações com atualização em tempo real para estações metroferroviárias. , [S.l.], 2021.

TANENBAUM, A.; WETHERALL, D. **Redes de computadores**. [S.l.]: Pearson Prentice Hall, 2011.

TRENSURB - Empresa de Trens Urbanos de Porto Alegre S.A. https://www.trensurb.gov.br/servicos/horarios, 2024.

TRENSURB. **Relatório Integrado 2022**. 2023.