Robotics and AI Capstone Design Report

Available Parking Lot Web Application

62011087 Ananchana Lertphitaksuntorn

62011171 Nipattra Tangsopa

62011268 Kritjuhn Kaklai

62011306 Thammachet Asavathongkul

Robotics and AI Engineering

School of Engineering

King Mongkut's Institute of Technology Ladkrabang

Academic Year 2022

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

PROJECT CERTIFICATE

Project Title:	Available Parking Lot Web Application
Students name:	Ananchana Lertphitaksuntorn
	Student ID 62011087
	Nipattra Tangsopa
	Student ID 62011171
	Kritjuhn Kaklai
	Student ID 62011268
	Thammachet Asavathongkul
	Student ID 62011306
Degree:	Bachelor of Engineering in Robotics and AI
Project Advisor:	Signed _____
	(Dr. Kanoknuch Songsuwankit)

Project Title: Available Parking Lot Web Application
Students name: 62011087 Ananchana Lertphitaksuntorn
62011171 Nipattra Tangsopa
62011268 Kritjuhn Kaklai
62011306 Thammachet Asavathongkul
Faculty: School of Engineering
Degree: Bachelor of Engineering in Robotics and AI Engineering
Project Advisor: Dr. Kanoknuch Songsuwankit

ABSTRACT

ParkShare is an innovative web application designed to address the challenge of finding available parking spaces in urban areas. Leveraging the power of geolocation, real-time data, and artificial intelligence (AI), ParkShare provides users with up-to-date information about parking availability. The application uses an IP camera with Faster RCNN ResNet50 Tensorflow pretrained model for car detection and sensor technology by using TCRT5000 with ESP32 microprocessor, ensuring accurate and timely updates.

A distinguishing feature of ParkShare is its AI-powered chatbot assistant, which facilitates seamless communication between users and the system. The chatbot, built on natural language processing and machine learning techniques, which is MongoDB for data collection and Cosine Similarity as a searching tool, is capable of understanding user queries and providing relevant information about parking spaces and surrounding areas.

Additionally, ParkShare features an intuitive, easy-to-read map interface with color-coded pins representing parking availability. Users can search for parking spaces in their vicinity or at specific locations.

In conclusion, IR sensor parking model, ESP32 can receive the updated data from IR sensor and send to the server within the 1 second delayed time. The result of IP camera AI detection. The detection accuracy at daytime is around 100%, and at nighttime is around 71%.

Keywords: *IP Camera, Web application, Detection, Recognition, AI, Chatbot, Real-time data collection, Faster RCNN ResNet50, Cosine Similarity, TCRT5000, ESP32*

ACKNOWLEDGEMENT

We are deeply grateful for the support and guidance we have received during the creation of this academic report. The journey was both enlightening and challenging, made possible through the contributions of many.

First and foremost, we would like to express our sincere gratitude to our research advisor for their unwavering support and invaluable guidance. Their insightful feedback and thought-provoking questions have shaped our research and intellectual growth.

We are immensely grateful to our colleagues and peers, who have been a source of inspiration and camaraderie throughout this journey. Their critical questions, stimulating discussions, and constructive feedback have been integral to the development and refinement of our research.

To our family and friends, your unconditional support and encouragement have been our pillar of strength. You're understanding and patience during the late-night study sessions and missed social events did not go unnoticed or unappreciated.

Lastly, we would like to acknowledge the importance of maintaining a balanced lifestyle during this process. To the local coffee shop baristas, thank you for providing a warm and welcoming environment to work and reflect.

In conclusion, the creation of this academic report has been a collective endeavor. It represents not only our efforts but the support, guidance, and encouragement of many individuals. We are eternally grateful for your contributions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

1. Background and significance

In recent years, urbanization and population growth have led to an increasing demand for public parks and recreational spaces in cities. Finding an available park in densely populated areas can be burdensome, and traditional methods of searching for parks may not provide real-time information about their occupancy status, the current method only shows the available park real-time at those places.

To address this issue, a park sharing web application has been developed to simplify the process of discovering empty parks while enhancing user experience through AI-powered chatbot assistance. ParkShare is a web-based application that addresses the need for user-friendly and effective solutions for locating the available park spaces in real-time. It aims to enhance the overall park experience by providing users with accurate and up-to-date information on park availability.

Public parks and recreational areas are in higher demand in cities now than they were a few years ago due to urbanization and population expansion. It can be challenging to find a park that is open in highly crowded areas, and conventional park search methods may not provide real-time information about their occupancy status; the present method only displays the open park in those locations in real-time.

To solve this problem, a park sharing web application has been created for the user to easily find unoccupied parks and improves user experience with chatbot support powered by AI. ParkShare is a web-based program that responds to the demand for simple and efficient ways to find open park spots in real-time. It attempts to improve visitors' overall park experiences by giving precise and up-to-date information.

2. Objectives

2.1 To satisfy the users using the application, ParkShare.

2.2 Be able to detect the car by the IP camera and send it to the central server.

2.3 Be able to send the information from the sensor and send it to the central server.

2.4 Be able to display the availability of the parking lot on the applications.

3. Scope of the project

3.1. Application Features:

a. Real-time Park Lot Availability: The application provides users with live updates on the availability of parking lots at the HM building mock-up park lot and the ATROBOT store, leveraging IR sensors and image processing technology, respectively.

b. User Interface: A user-friendly interface has developed to ensure intuitive navigation and a seamless user experience.

c. Map Integration: The application incorporates with a map view, enabling users to visualize the locations of the parking lots and their respective availability.

d. Search Functionality: Users have been able to search for specific locations or addresses to quickly access the parking lot availability information.

e. Chatbot: Customers could engage in a live chat with a representative from the shop. This feature allows for extended interaction, providing customers with the convenience of communicating directly with a knowledgeable and helpful representative.

3.2 Data collection and Integration:

a. HM Building Mock-up Park Lot: The application utilizes IR sensors installed in the HM building mock-up park lot to collect real-time data on park lot occupancy.

b. ATROBOT Store: AI image processing technology be employed to gather real-time park lot occupancy data at the ATROBOT store.

c. Data Integration: The application integrates with the respective sensor systems to fetch and update the park lot availability data also with camera.

4. Method of conducting the project

4.1. Project Planning

This phase involves defining the project's objectives, scope, and features. A preliminary study was conducted to understand the domain, identify the problem, and propose an efficient solution. The technology stack to be used, hardware components, and AI technologies for image detection were decided upon during this phase.

4.2. System Design

Once the project scope was defined, the next step was to design the application. This involved mapping out the system architecture and designing the hardware components. The system architecture includes the sensor parking area model, IP camera scenario, backend, and frontend. The sensor hardware components (TCRT5000 and ESP32) were also planned for in this stage.

4.3. Implementation

The implementation phase involved building the application according to the designs. The sensor parking area model was created with TCRT5000 and ESP32. An IP camera scenario was set up. The backend of the application was developed using Falcon API, MongoDB, Nginx, Gunicorn, and was hosted on XCP-NG. The frontend was built using React JS. Integration of the sensor and camera data was also done during this phase.

4.4. Testing

After the implementation, extensive testing was carried out to ensure the accuracy and efficiency of the application. Testing was performed at different levels of the system, including the sensor parking area model, IP camera scenario, backend, and frontend, to ensure the system was functioning as expected. The F_1 score was used to evaluate the performance of the AI image detection system.

4.5. Deployment

Post successful testing, the application was deployed. The backend was deployed using XCP-NG as the operating system. For the frontend, a suitable hosting platform was chosen based on the scalability, performance, and cost requirements.

4.6. Maintenance and Updates

After the deployment, the application was closely monitored to ensure it was functioning optimally. Regular maintenance and updates were scheduled to improve the application based on user feedback and technological advancements.

5. Outcome

5.1. Efficient Real-time Parking Information

Utilizing sensor technology and AI image detection, the application provides users with real-time parking space availability. This feature will eliminate the need for manual search, saving users time and reducing traffic congestion. This outcome aligns with the theories of AI Image Detection and Technology of Sensor.

5.2. Accurate Parking Positioning

Leveraging the Geographic Coordinate System, the application offers precise parking location data. This feature will improve the parking experience by offering users accurate directions to available parking spots.

5.3. Improved User Interface (UI) & Experience (UX)

The Human-Computer Interaction (HCI) theories, such as Cognitive Load Theory, Fitts' Law, and Distributed Cognition, guide the application's interface design. The application will have an intuitive and user-friendly interface, minimizing cognitive load, and making navigation easier. This will significantly enhance the user experience.

5.4. Scalable and Robust Architecture

The use of virtual machines and effective network communication protocols will result in a scalable and robust application. The application will be capable of handling increased user traffic and maintaining high performance even under strain.

5.5. Privacy and Security

By adhering to best practices in database management and server management, the application ensures data privacy and security. Users' personal information and usage data will be well-protected.

Chapter 2 Concepts, Theories, and Related Resource

1. Theory

1.1. Technology of Sensor

These technologies play a significant role in object detection, translating physical phenomena such as motion or light into data signals. In this project, sensors are utilized to determine whether a parking spot is free or occupied. To specifically detect the presence of a car, sensors can be installed in parking places.

Infrared Sensor Technology

Figure 1 IR Emitter and Receiver

An infrared (IR) sensor functions as an electronic apparatus designed to gauge and identify the presence of infrared radiation within its immediate surroundings.

Its purpose lies in the ability to accurately measure and detect the infrared emanating from various sources in the sensor's vicinity. By utilizing this technology, the IR sensor serves as a reliable means to assess and capture infrared radiation, enabling applications in diverse fields.

In the world of infrared sensors there are two primary types: active and passive. Active infrared sensors possess the dual capability of emitting and detecting infrared radiation. These sensors consist of two components: a light emitting diode (LED) and a receiver as shown in figure 2. As an object near the sensor, the LED emits infrared light that subsequently reflects off the object's surface. This reflected infrared light is then captured by the receiver, enabling the sensor to discern the presence of the object as shown in figure 3. Active infrared sensors find widespread

application as proximity sensors, commonly employed in obstacle detection systems, such as those utilized in robots. In essence, served the purpose of identifying objects in close range and aiding in the navigation around potential obstacles. As in figures 2 and 3 show how the infrared light bounces to the receiver when there are objects and no object.

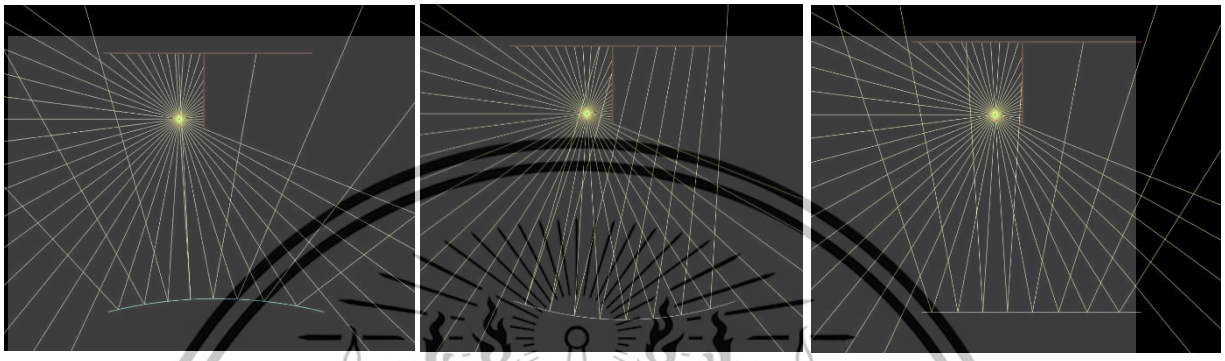


Figure 2 IR sensor when there are objects near receiver (emitter) on the left-side

Figure 3 IR sensor at normal state (emitter on the left-side)

Passive infrared (PIR) sensors are specifically designed to detect infrared radiation without emitting it using an LED. These sensors rely on their ability to capture the natural infrared radiation emitted by objects and living organisms in their surroundings. When an object moves within the sensor's field of view, it detects the fluctuations in infrared radiation patterns caused by the object's heat signature. This detection triggers a response, often utilized to activate or control various devices and systems such as security systems, lighting fixtures, or automated doors.

In simpler terms, PIR sensors operate as "motion detectors" that respond to alterations in the infrared energy present in their environment. Unlike active infrared sensors that emit infrared light to detect objects, PIR sensors passively monitor the heat emitted by objects without emitting any light themselves. Whenever a warm object enters the sensor's range and produces a change in the detected infrared

radiation, the PIR sensor identifies this variation and can initiate a programmed action or generate an alert. This technology finds wide application in numerous fields, including home security systems, energy-efficient lighting systems, and occupancy sensing systems.

Infrared light as in figure 4, a component of the electromagnetic spectrum that is invisible to the human eye, can be utilized effectively for vehicle parking detection without causing any disturbance to the user. A crucial advantage of using infrared light is that its wavelengths are beyond the spectrum visible to humans, ensuring that the user will not be disturbed by any apparent flashes or emissions. Furthermore, infrared light is not impacted by ambient light conditions, providing reliable performance day or night. [1]

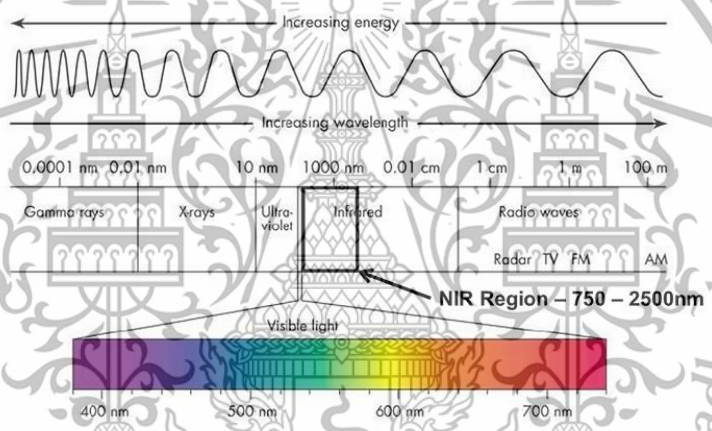


Figure 4 Wavelength

1.2. AI Image Detection

A subfield of computer science called artificial intelligence (AI) is concerned with building intelligent machines that can carry out tasks that ordinarily call for human intelligence. AI systems are made to perceive their surroundings, think about them, and then decide what to do or not do to accomplish certain objectives. Machine learning and deep learning are two types of AI. This project, which employs object detection to find and count the automobiles in an outdoor parking lot, is a good fit for deep learning [2].

1.2.1. Artificial Neural Networks or Neural Networks

Neural Network or Artificial Neural Network is a type of AI machine learning, which is called Deep learning. A node layer of an artificial neural network (ANN) consists of an input layer, one or more hidden layers, and an output layer, as shown in figure 5. Each node, or artificial neuron, is connected to others and has a weight and threshold that go along with it. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer. Otherwise, no data is transmitted to the network's next tier.

In a traditional Neural Network (NN), the architecture, as shown in figure 5, consists of multiple layers of interconnected nodes, or neurons. Each layer is responsible for a different stage of data processing, and together they enable the network to make complex predictions or decisions based on input data [3].

The types of layers in a basic Neural Network include:

Input Layer: The input layer is the initial stage in the architecture of a neural network. It's the point of contact where the network receives information to be processed. Each node in the input layer represents one feature or dimension of the input data. For instance, in an image recognition task, the input layer might include as many nodes as there are pixels in the input images.

Hidden Layers: Hidden layers reside between the input and output layers and are responsible for most of the computation performed by the network. Each hidden layer applies weights to the inputs and passes them through an activation function. The output is then sent forward to the next layer.

In the context of Deep Learning, a neural network can consist of many hidden layers, hence the term "deep". The deep architecture allows the model to learn hierarchical representations of the input data. For example, in image recognition, early hidden layers may detect edges and

colors, while deeper layers might recognize more complex shapes or objects.

Output Layer: The output layer is the final layer in a neural network. It takes the values computed by the hidden layers and transforms them into the final output values. The number of nodes in the output layer depends on the type of task - for binary classification, there is typically a single output node, while for multi-class classification, there might be as many output nodes as there are classes.

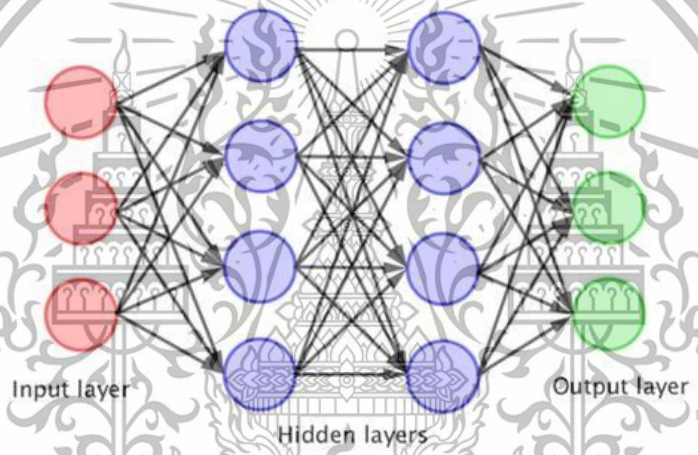


Figure 5 Layers in Neural Networks

Each layer within a neural network is made up of individual computational units known as perceptron, or artificial neurons. These perceptron's are the fundamental building blocks of the network. They each embody three core elements - weights, biases, and an activation function.

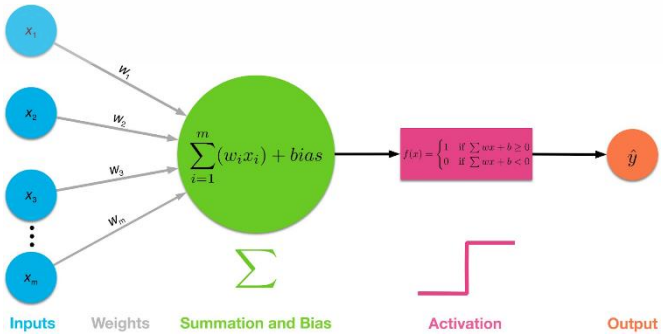


Figure 6 Perceptron Network Perceptron

The activation function plays a critical role in each perceptron. It determines the output a neuron will produce based on the input it has received. Essentially, the activation function translates the inputs, passed through the weighted sum and bias, into an output that will be either transmitted to the next layer or become the final output of the network. There are various types of activation functions used in neural networks including this function.

Sigmoid: The sigmoid function, also known as the logistic function, is a type of activation function that is traditionally used in neural networks. It takes a real-valued number and squashes it into a range between 0 and 1, as shown in Figure 7. This can be useful for output neurons in binary classification problems, where we want to interpret the output as a probability.

The mathematical representation is:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \text{--- (1)}$$

where x is the input to the function, and e is Euler's number, the base of the natural logarithms. While sigmoid functions have been widely used in the past, they are less commonly used now due to issues such as vanishing gradients, which can slow down training.

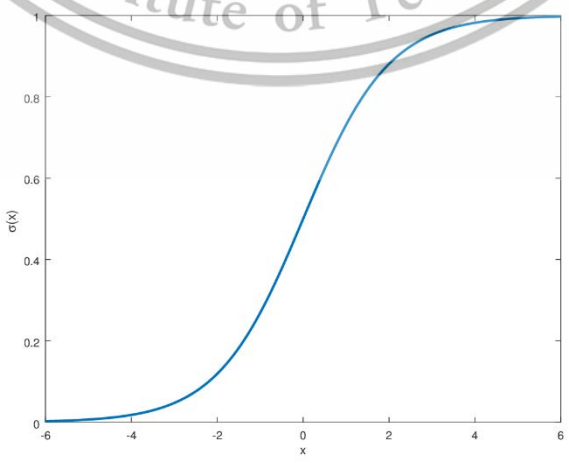


Figure 7 The output shape of the Sigmoid function

ReLU (Rectified Linear Unit): The ReLU function has become very popular in the last few years. It computes the function, $f(x) = \max(0, x)$. In other words, the output is the input directly if it is positive, else it's zero.

The mathematical representation is:

$$f(x) = \max(0, x) \quad \text{--- (2)}$$

ReLU is simple and efficient and has been shown to speed up training. However, it can suffer from the dying ReLU problem, where neurons can sometimes get stuck during training and stop updating, especially when dealing with high learning rates [3].

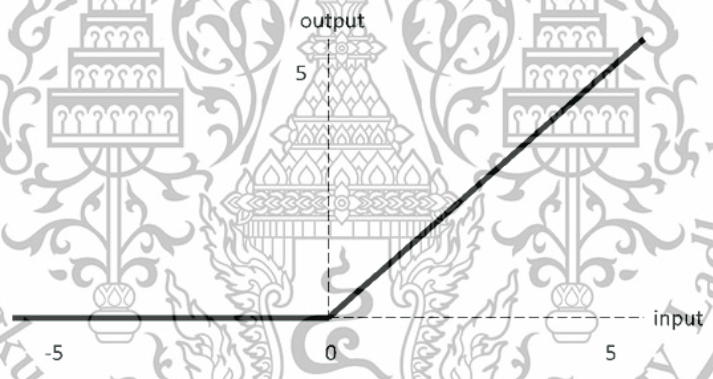


Figure 8 The output shape of the ReLU Function

1.2.2. Convolutional Neural Networks (CNN)

Convolutional Neural Networks known as CNN or ConvNet is a deep learning algorithm that is better for use in computer vision to analyze the images data, e.g., Face detection, Image classification, Object detection, etc.

CNN consists of pooling layer, Convolutional layer, fully connected layer [3].

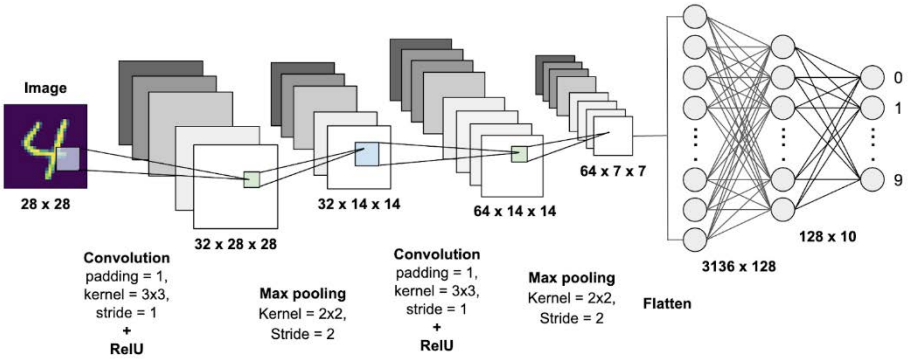


Figure 9 Convolutional Neural Networks Architecture

CNNs utilize specialized layers to optimize their understanding of image data [5].

Convolutional Layer: The primary layer that gives Convolutional Networks their name. Here, several filters (or kernels) are applied to the input data. These filters are small matrices that slide across the input data and perform element-wise multiplication followed by a summation, forming a feature map.

Pooling Layer: Also known as a down sampling layer, its function is to reduce the spatial dimensions (i.e., width and height) of the input data, preserving the depth. Two common types of pooling:

Max Pooling: This is a type of pooling operation that involves selecting the maximum value from each of a series of sub-regions of the input. It's particularly effective when the input contains features that vary in intensity. By selecting the maximum value from each sub-region, max pooling ensures that these distinctive, intense features are captured and passed on to subsequent layers, irrespective of their location in the sub-region.

The operation can be represented as follows:

If P is the pooling function, and X represents a sub-region of the input then,

$$P(X) = \max(X) \quad \text{--- (3)}$$

In other words, for each sub-region X in the input, the maximum value within X is taken as the output for that region.

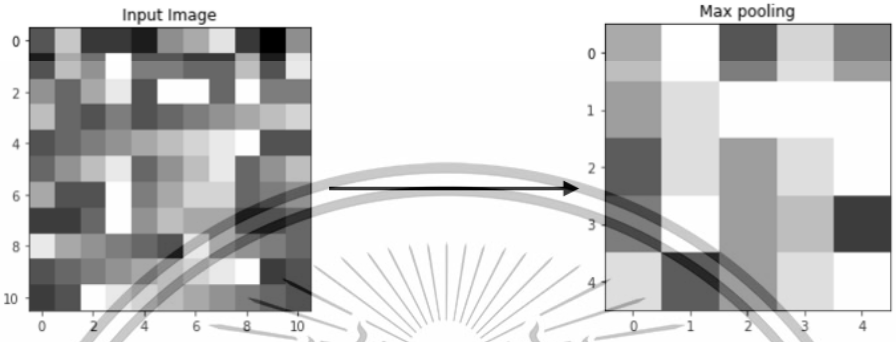


Figure 10 Result all Max Pooling, Lighter color indicate higher value.

Average Pooling: In contrast to max pooling, average pooling calculates the average value of each of a series of sub-regions of the input. As in figure 10, this method equally considers all features within each sub-region, not just the most intense ones. It can sometimes provide a more balanced representation of the input data than max pooling, especially when all features are equally important.

The operation can be represented as follows:

If P is the pooling function, and X represents a sub-region of the input, then,

$$P(X) = avg(X) \quad \text{--- (4)}$$

That can be seen for each sub-region X in the input, the average value within X is computed and used as the output for that region.

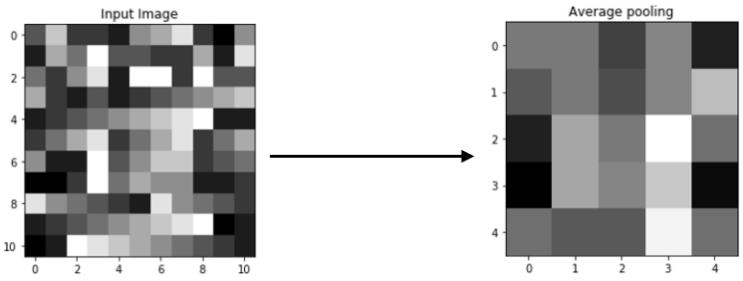


Figure 11 Result of Avg Pooling

Fully Connected Layer: Also known as a dense layer, each neuron in a fully connected layer is connected to every neuron in the previous layer. If the previous layer's output is Y and the weight matrix for the current layer is W, the bias is b, and the activation function is f, the output Z of the fully connected layer can be represented as:

$$Z = f(Y \times W + b) \quad \text{--- (5)}$$

Normalization Layer: Layers like Local Response Normalization (LRN) or Batch Normalization normalize the activations of the neurons, which can speed up learning and improve the general performance of the model. In batch normalization, if X is the input data, the output Y is given by:

$$Y = \gamma \times (X - \mu)/\sqrt{\sigma^2 + \varepsilon} + \beta \quad \text{--- (6)}$$

Where γ and β are parameters learned during training, μ and σ^2 are the mean and variance of the batch, and ε is a small constant for numerical stability.

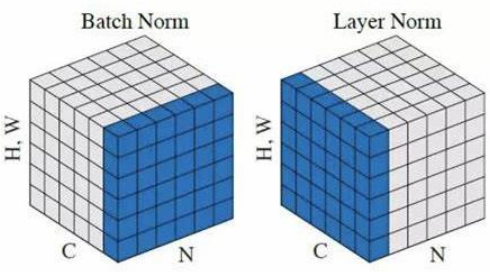


Figure 12 Batch Norm and Layer Norm

Dropout Layer: It randomly shuts off some neurons in the layer during training, preventing overfitting. This is a form of regularization. If the input to the layer is X, the output Y is:

$$Y = X \times M \quad \text{--- (7)}$$

Where M is a binary mask where each element is 0 with probability p (the dropout rate) and 1 otherwise. [2]

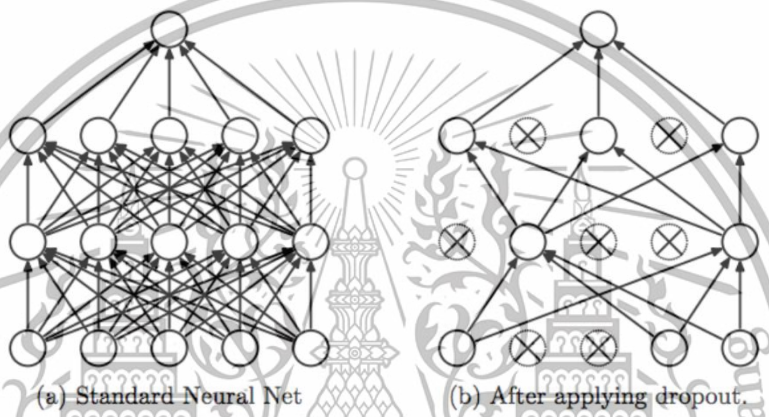


Figure 13 Neural Networks (RCNN)

1.2.3. Region-based Convolutional Neural Networks (RCNN)

RCNN is developed from CNN, but uses the CNN technique only for Feature Extraction, then uses SVM (Support Vector Machine – analyzes data for classification and regression analysis) to classify the class of object. This type of neural network introduced the technique called Region Proposal Technique. The disadvantage is that each warped region in RCNN needs to pass through the Convolutional Neural Network one-by-one. This makes RCNN very slow in detection [4] [6].

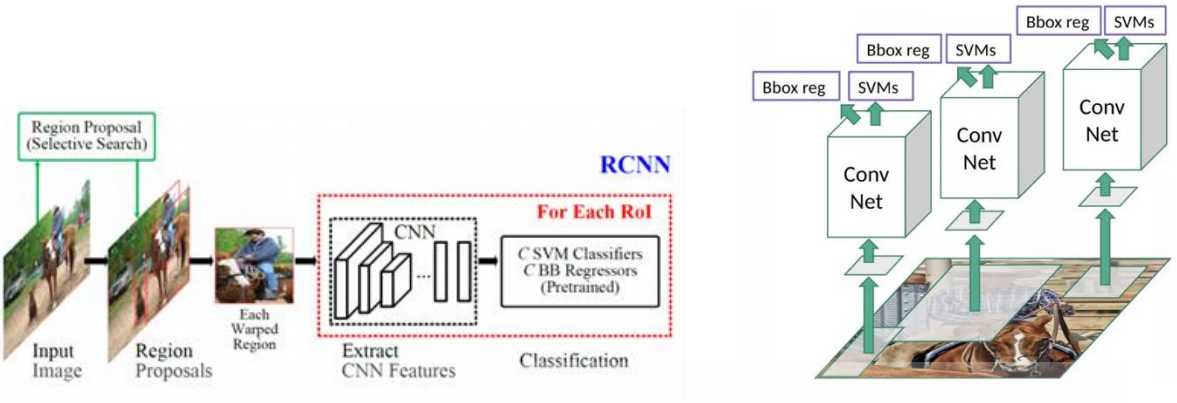


Figure 14 Region-based Convolutional Neural Networks Architecture

As shown in figure 14, the wrapped region passing through the process of ConvNet, then the classify output display. Then, the next warped region is also passing through the ConvNet to classify the object, and the process of RCNN will continue to do the same thing. This is the reason for the delay speed in detection.

Region proposals: Region proposals are simply the smaller regions of the image that possibly contains the objects we are searching for in the input image. To reduce the region proposals in the R-CNN uses a greedy algorithm called selective search.

Selective Search: Selective search is a greedy algorithm that combines smaller segmented regions to generate region proposal. This algorithm takes an image as input and output generate region proposals on it.

SVM (Support Vector Machine): The feature vector generated by CNN is then consumed by the binary SVM which is trained on each class independently. This SVM model takes feature vector generated in previous CNN architecture and outputs a confidence score of the presence of an object in that region. However, there is an issue for training with SVM is that we required AlexNet feature vectors for training SVM class. So, we could not train AlexNet and SVM independently in parallel manner. This challenge is resolved in future versions of R-CNN (Fast R-CNN, Faster R-CNN etc.).

Bounding Box Regressor: In order to precisely locate the bounding box in the image.

1.2.4. Fast Region-based Convolutional Neural Networks (Fast RCNN)

Fast RCNN is an improvement of RCNN, it fixes the delay in the process of RCNN. Process the input image (whole image) through CNN, after that pull out the region. This can reduce the delay in reprocessing, instead of reprocessing every time, this uses only one time input of the whole image [6].

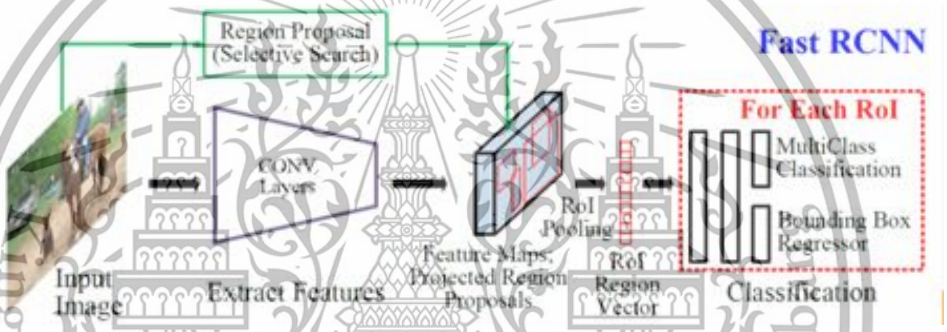


Figure 15 Fast Region-based Convolutional Neural Networks Architecture

1.2.5. Faster Region-based Convolutional Neural Networks (Faster RCNN)

Faster RCNN is designed to improve the detection speed of normal RCNN and Fast RCNN.

Faster RCNN was invited to improve the limitations and the detection speed of RCNN, and Fast RCNN. The key point of Faster RCNN is to include Region Proposal Network (RPN) that generates potential object bounding box proposals in an efficient manner. The RPN is a convolutional neural network that takes an image as input and outputs a set of object proposals along with their corresponding objectness scores [6] [9].

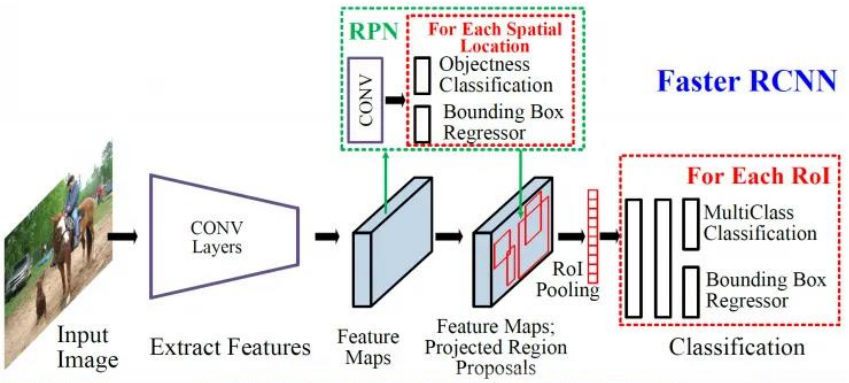


Figure 16 Faster Region-based Convolutional Neural Networks Architecture

Working steps of Faster RCNN

1. Input image: The algorithm takes an input image and passes it through a convolutional neural network (CNN) to extract a set of feature maps. These feature maps capture the visual content of the image at different scales.

2. Region Proposal Network (RPN): The RPN takes the feature maps as input and generates a set of potential objects bounding box proposals. It achieves this by sliding a small window (called an anchor) over the feature maps at different positions and scales. For each anchor, the RPN predicts two things: the probability of the anchor containing an object and the coordinates for refining the anchor to better align with the object (bounding box regression).

3. Region of Interest (RoI) Pooling: The proposed bounding box regions from the RPN are combined into a set of fixed-sized regions of interest. These regions are aligned and cropped from the feature maps using a technique called RoI pooling, which converts them into a fixed size for further processing.

4. Region Classification and Bounding Box Regression: The fixed-sized regions of interest are fed into a classifier network and a regressor network. The classifier network assigns a class label to each region (e.g., person, car, etc.), and the regressor network refines the coordinates of the bounding box proposals.

5. Non-Maximum Suppression (NMS): The bounding box proposals are filtered based on their object scores and refined using the regressor

network. Non-maximum suppression is applied to remove duplicate detections and select the most confident detections for each object.

The main point that makes Faster RCNN better than the other 2 RCNN is Region Proposal Network (RPN).

Region Proposal Network (RPN) 4is to generate potential object bounding box proposals in an efficient and effective manner. It achieves this by examining different regions of an input image and determining their likelihood of containing an object.

Region Proposal Network (RPN)

Region Proposal Network (RPN) is to generate potential object bounding box proposals in an efficient and effective manner. It achieves this by examining different regions of an input image and determining their likelihood of containing an object.

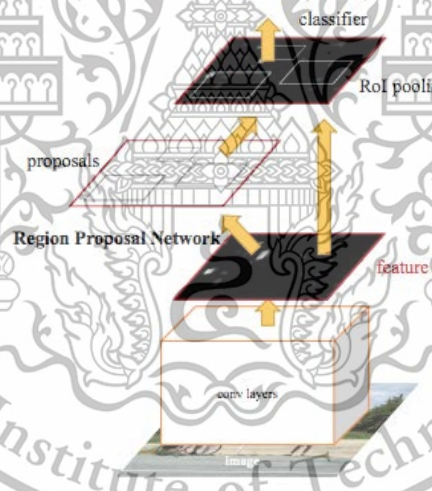


Figure 17 Region Proposal Network (RPN)

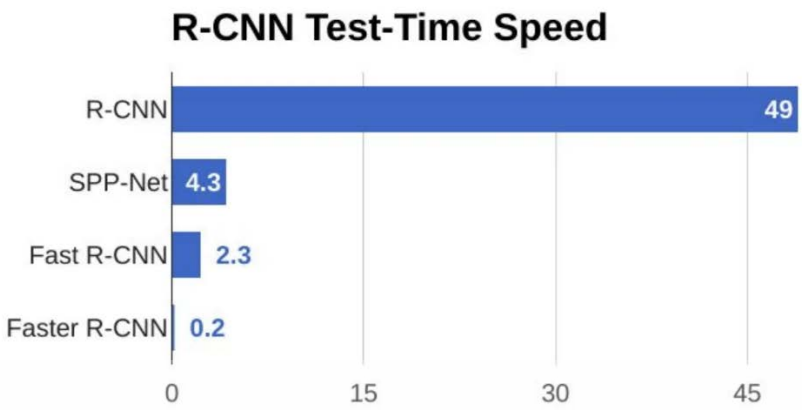


Figure 18 RCNN Test-Time Speed

As shown in Figure 18, the comparison between RCNN, Fast RCNN, and Faster RCNN, the Faster RCNN process faster than RCNN by 99.6% and Fast RCNN by 91.3%. This is the reason why Faster RCNN is popular for real-time object detection.

1.2.6. ResNet50

ResNet50 is a residual network with 50 layers consisting of convolutional neural networks, which are convolutional layers, pooling layers, fully connected layers, and skipped connections. Skipped connections are known as residual connections.

The main idea of ResNet is to solve the vanishing gradients issue that arises during the training of extremely deep neural networks. Due to the challenges of deep model optimization, deep networks frequently experience degradation, wherein the addition of new layers results in performance reduction. In order to properly train and improve deep networks, ResNet introduces residual connections that enable the network to learn residual mappings [7].

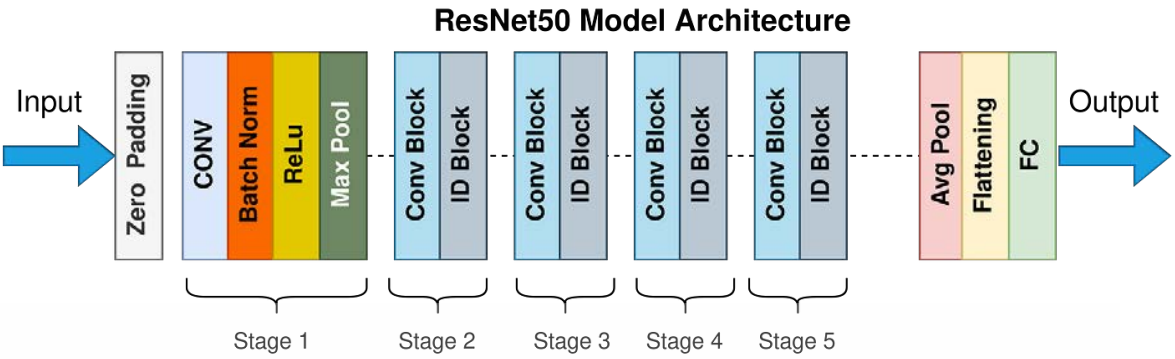


Figure 19 ResNet50 architecture

In figure 19, each residual block in the ResNet-50 architecture has many convolutional layers. As the network gets deeper, these blocks capture features that are more abstract. By allowing gradients to pass through the network more directly, skip connections solve the vanishing gradient problem and make it possible to train very deep models quickly.

1.2.7. Object detection

Object detection is a computer vision and image processing technique for locating and specifying the object that caught on the image data. There are many types of AI models for detection but in this case, Faster RCNN is the most accurate.

1.2.8. F1 Score

The F1 score is a metric used to assess the performance of a classification model, particularly in cases where there is an imbalance between the classes or when both precision and recall are important [10].

Precision measures the accuracy of positive predictions, indicating the proportion of correctly predicted positive instances out of all instances predicted as positive. It focuses on minimizing false positives.

Recall, also known as sensitivity, measures the model's ability to identify positive instances by calculating the proportion of correctly

predicted positive instances out of all actual positive instances. It focuses on minimizing false negatives.

The F1 score is the harmonic mean of precision and recall. By considering both precision and recall, the F1 score provides a balanced evaluation of the model's effectiveness, especially when there is an uneven distribution between the classes.

To calculate the F1 score, you first compute precision and recall using the following formulas:

$$Precision = \frac{True\ Positives}{(True\ Positives + False\ Positives)} \quad \text{--- (8)}$$

$$Recall = \frac{True\ Positives}{(True\ Positives + False\ Negatives)} \quad \text{--- (9)}$$

Which:

True Positives (TP) means an outcome where the model correctly predicts the positive class.

True Negative (TN) means an outcome where the model correctly predicts the negative class.

False Positive (FP) means an outcome where the model incorrectly predicts the positive class.

False Negative (FN) means an outcome where the model incorrectly predicts the negative class.

Then, the F1 score is computed as follows:

$$F1 = \frac{2 \times (Precision \times Recall)}{(Precision + Recall)} \quad \text{--- (10)}$$

The F1 score ranges from 0 to 1, with 1 indicating perfect precision and recall, and 0 indicating the worst performance.

In summary, the F1 score considers both precision and recall, making it a valuable metric for evaluating classification models, especially when

there is class imbalance or when both precision and recall are important factors to consider.

1.3. Positioning Theory

1.3.1. Geographic coordinate system (GCS)

The Geographic Coordinate System (GCS) is essentially a grid of references that allows for the precise pinpointing of any location on the Earth's surface. It operates based on two fundamental coordinates, namely latitude and longitude. Latitude lines, with the equator at zero degrees, circle the globe horizontally, while longitude lines, with the Prime Meridian as the zero-degree reference, encircle it vertically. Each location on Earth is thereby determined by its unique angular distance from these two foundational lines, providing a universally accepted mechanism for specifying any position.

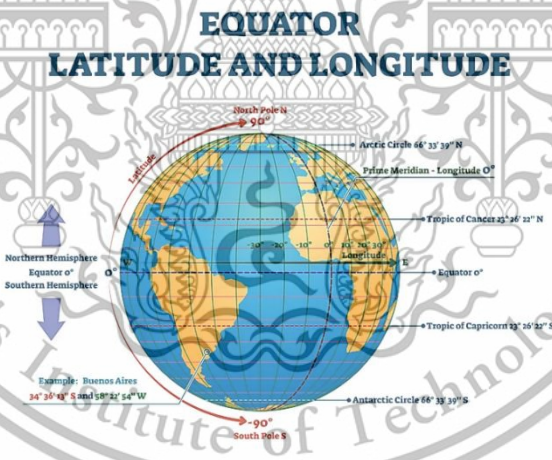


Figure 20 Latitude and Longitude System

1.4. Human-Computer Interaction (HCI)

HCI is a field dedicated to understanding the interaction between humans and computers, and to designing for successful human use. In this case, it pertains to front-end design. HCI principles influence the design of a user interface to ensure a smooth interaction flow and a user-friendly experience. [3]

1.4.1. Cognitive Load Theory

The Cognitive Load Theory underscores that the complexity of a digital interface impacts its usability. It asserts that if an interface places an excessive cognitive burden on a user, their interaction and learning efficiency can be compromised due to our cognitive processing capacity being finite. The theory breaks down this cognitive load into three categories: intrinsic load, related to the inherent complexity of the content; extraneous load, linked to how the content is delivered; and germane load, associated with the cognitive effort invested in comprehending the system. Therefore, designers striving for optimal user experiences must strike a balance between these aspects, ensuring that their designs are not cognitively overwhelming but rather facilitate intuitive understanding and interaction. [4]

1.4.2. Fitts' Law

In the context of HCI, Fitts' Law is a crucial concept that delineates the correlation between the duration it takes for an individual to aim at a specific target and the target's distance and size. The law essentially posits that the time consumed to reach a target depends on its distance and its size. This concept is extensively leveraged in various facets of user interface design to improve the user experience, by strategically adjusting the size and location of interactive components.

Fitts's Law is a mathematical model that predicts the time required for a human operator to move a pointer or cursor to a target area. It was proposed by psychologist Paul Fitts in the 1950s and is significant for various fields including human-computer interaction (HCI), ergonomics, and user interface design.

The law is mathematically defined as follows:

$$T = a + b\ log_2 \frac{2D}{w} \quad \text{--- (11)}$$

Where:

T is the time taken to complete the movement.

D is the distance from the starting point to the center of the target.

W is the width of the target along the axis of motion.

a and b are empirically determined constants, with a being the intercept and b being the slope of the function.

The log_2 term is the Index of Difficulty (ID) and is expressed in bits.

The term $log_2(\frac{2D}{W})$, often referred to as the Index of Difficulty (ID), captures the trade-off between distance and size of the target. A larger distance or smaller target size results in a higher Index of Difficulty, thus requiring more time for the movement.

Key insights from Fitts's Law can be summarized as follows:

The time required to move to a target increases with the distance to the target. Closer targets are thus quicker to acquire.

The time required to move to a target decreases with the size of the target. Larger targets are thus quicker to acquire.

This relationship, however, is not linear but logarithmic, indicating that time taken to reach a target increases at a decreasing rate as the Index of Difficulty increases.

Fitts's Law demonstrates that interaction speed is constrained by the spatial properties of the task, i.e., the distance and size of the target. This provides critical insights into the design of user interfaces and can guide the design of any interactive system, thereby promoting efficiency and ease of use. [5]

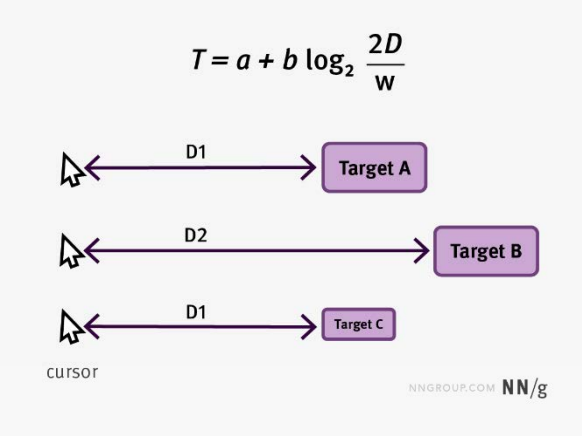


Figure 21 Fitts' Law Equation

1.4.3. Distributed Cognition

Distributed Cognition [6] in Human-Computer Interaction (HCI) posits that cognitive processes aren't confined within the boundaries of an individual but spread across individuals, objects, and technologies. This theory suggests that human cognition is essentially a system-level phenomenon, where the system comprises people, artifacts, and the surrounding environment. It challenges the traditional conception of cognition as an individual-centric process, instead arguing that cognitive tasks are accomplished through a complex interplay of mental and material resources distributed across space and time.

In the context of HCI, Distributed Cognition emphasizes the design of interfaces and technologies that align with the interconnected and collective nature of cognitive processes. It underscores the importance of considering the entire cognitive ecosystem while designing digital interfaces, emphasizing that a well-designed interface should facilitate the smooth flow of information and cognitive processes across the system's elements.

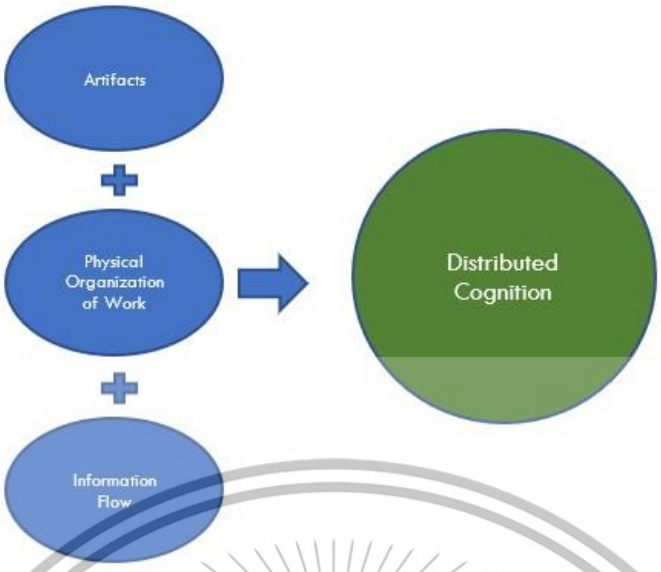


Figure 22 Distributed Cognition

Figure 22 depicts the tripartite principles integral to the notion of distributed cognition from the perspective of Human-Computer Interaction (HCI). When these principles function in isolation, distributed cognition is unable to manifest, primarily because there are no provisions for insights or inputs from an external source. Take, for instance, an HCI scenario wherein a user interacts with a digital interface for data analysis. The physical organization of the task is demonstrated, but without an exchange of information with another user or without documenting findings in a shared database, the environment for distributed cognition is not established. The amalgamation of all three principles gives rise to distributed cognition.

An illustrative example could be the introduction of a novel workflow in an HCI context. Here, multiple users may collaborate in a shared digital space, reviewing and modifying data simultaneously. Such a setting engenders distributed cognition.

In an HCI context, distributed cognition refers to a cognitive process that is not confined to an individual. Instead, it extends across individuals, artifacts, and representations, facilitated by the interactive and collaborative nature of digital systems. This leads to a more holistic understanding of the data, enhances problem-solving abilities, and improves decision-making. Therefore, it's crucial to promote such collaborative interactions in HCI design.

1.5. Network Communication

1.5.1 Hypertext Transfer Protocol (HTTP)

HTTP, or Hypertext Transfer Protocol, is arguably a cornerstone in the evolution of digital interaction and the contemporary internet. This protocol has revolutionized our engagement with digital data, paving the way for a more interlinked global society.

HTTP works by operating as a request-response mechanism within the client-server computing architecture. In this scenario, a client (such as a web browser) sends an HTTP request to a server (which could be an application on a computer hosting a website). The server then responds to this request, usually with a web page, although it could also provide image data, a file for download, or simply an empty response. This interactive exchange is conducted over TCP/IP connections.

For all the network communication in this project send the payload with HTTP. For ESP32, after getting the update from the IR sensor, POST request is used to send the information to the server via the WiFi which also the client-server communication. For object detection by IP camera, after getting the update from the camera, use URL of the server to send the information to the server. [7]

1.5.1.1 HTTP Methods

a. GET

This verb is utilized for pulling data from the server. It solicits a rendition of a specific resource, and if available, the server transmits it back to the client.

b. POST

This verb is employed to transmit data to the server to generate a new resource. The data dispatched to the server with a POST request is secured in the request body of the HTTP request.

c. PUT

This verb is utilized to refresh an existing resource with new data. The server ensures that the updated resource is preserved at the URI provided by the client.

d. DELETE

This verb is employed to eradicate a resource pinpointed by a URI. Upon successful elimination, the server usually returns a 200 (OK) status code.

e. HEAD

This verb is used to only fetch the headers of a resource, which houses information about the resource, rather than the resource itself.

f. OPTIONS

The OPTIONS method is a feature of HTTP that outlines the communication possibilities for a targeted resource. It allows a client to know which methods (GET, POST, HEAD, PUT, DELETE, etc.) are acceptable by the server for a given endpoint. It also can serve to verify if a server is operational. When receiving the OPTIONS request, the server will provide a response with headers containing information about the HTTP methods available for the specific URL. [8]

1.5.2. REST Architectural Style

REST (Representational State Transfer) is an architectural style used in web service development. It's centered on the utilization of stateless, client-server communication protocol, typically HTTP. RESTful systems are designed to be scalable, fast, and reliable, with components that can be managed and updated independently. The architectural style uses standard methods and media types, and communicates via a uniform interface, which

simplifies and decouples the architecture, allowing each part to evolve independently. The central concept of REST is the resource, which is accessed via its URL, and can be manipulated using the standard HTTP methods such as GET, POST, PUT, DELETE, etc. Representations of resources are usually in XML or JSON format [9] [10].

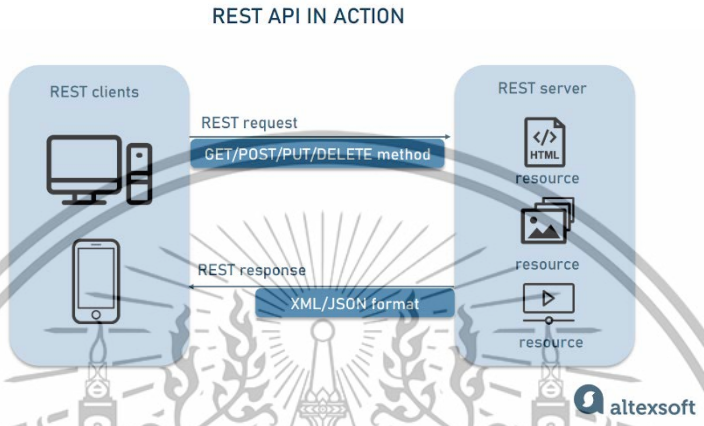


Figure 23 Rest API in action

1.6. Virtual Machines (VMs)

Virtual machines (VMs) represent a digital emulation of a physical computer, enabling an operating system to operate on top of the hardware of a physical device. They serve numerous purposes, including use in data centers and for testing software. [11]

1.6.1. Bare-Metal Virtual Machines or Type-1 or Native Hypervisors

The Bare-Metal Virtual Machines directly on the host's hardware, controlling it and managing the guest operating systems. The hypervisor here has unrestricted access to the hardware resources, which enhances performance, reliability, and scalability. Examples of such VMs include VMware ESXi, Microsoft Hyper-V, and Citrix XenServer.

Bare-metal VMs are predominantly used in enterprise-grade settings where performance and stability are paramount, such as data centers. Their direct interaction with the hardware enables higher performance and reduces susceptibility to slowdowns caused by other operating systems.

1.6.2 Hosted Virtual Machines or Type-2 or Hosted hypervisors.

Hosted Virtual Machines are hosted VMs operating on a traditional operating system like any other software program. Here, the hypervisor is a separate software layer installed on top of the host operating system. The hypervisor then manages calls for resources like CPU, memory, disk I/O, network, etc., through the underlying host operating system. Examples of hosted VMs include VMware Workstation, Oracle VirtualBox, and Parallels Desktop.

Hosted VMs are commonly used for consumer applications or in environments where the underlying hardware is not as vital. They offer greater flexibility as they can operate alongside other applications on a standard operating system.

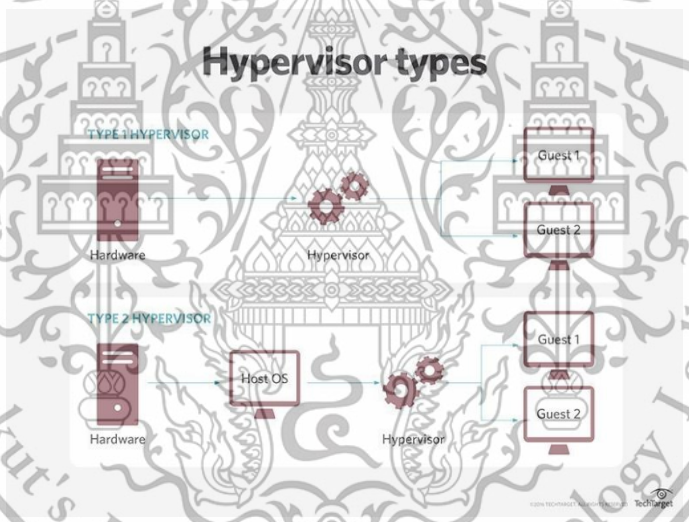


Figure 24 Type 1 and Type 2 hypervisor structure

Chapter 3 Material and Method

1. Problem statement

Given the issue of limited car park availability at the location, the primary concern arises from the difficulty experienced in locating an accessible parking space. The solution can vary depending on the specific situation and environment and may involve considering multiple factors before arriving at a decision.

In the first scenario where the location is equipped solely with a sensor, the gathered data is transmitted to our server for the purpose of presenting the current parking lot availability to the user. The methodology for detecting vehicle occupancy in shared parking systems involves the use of sensors and microcontrollers to accurately determine the availability of vehicles in parking spaces. Specifically, the TCRT5000 sensor and ESP32 microcontroller will be utilized to detect changes in light reflection. To implement this, the TCRT5000 sensor will be placed in a suitable location to detect the presence of vehicles within the parking space. The ESP32 microcontroller will receive and process the signal from the sensor, updating the central server with real-time occupancy status information via WIFI. These functions achieved the coding developed by using the Arduino app, with the code setup including sensor configuration and communication with the central server. This approach ensures the efficient and reliable detection of vehicle occupancy in shared parking systems, enabling users to locate available parking spaces quickly and easily.

The second approach utilizes AI image detection to count parked vehicles in outdoor areas where no physical sensors are present. This method involves capturing images through IP cameras, which are then subjected to AI processing using a pre-trained TensorFlow model that is specifically designed to identify and count cars. The model detects the number of cars present in the image and this information is displayed in a web application.

2. Sensor Parking Area Model

2.1. Hardware

2.1.1 TCRT5000

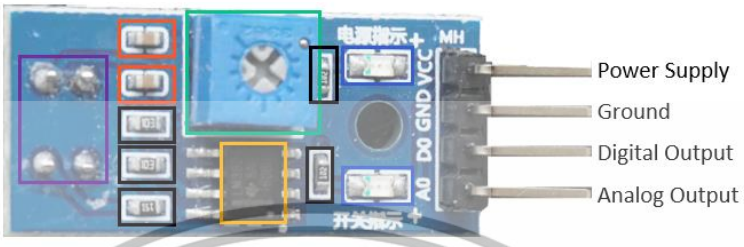


Figure 25 TCRT5000 Components

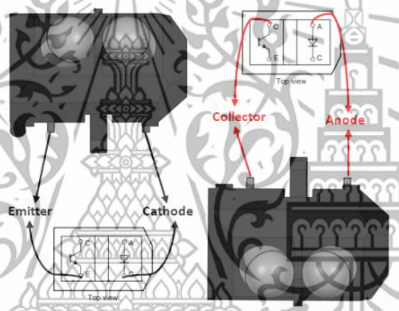


Figure 26 Physical of IR sensor (TCRT5000)

The TCRT5000 infrared sensor, illustrated in figure 25 and figure 26 as a part of the sensor, uses a phototransistor to detect changes in reflected light. Consisted of an infrared emitter and a phototransistor working to determine the presence of an object as demo cars shown in these projects. Each component of the TCRT5000 is labeled with distinct colors that can be seen in figure 25 indicating the Potentiometer (green), LM393 dual comparator (orange), Capacitors (red), Resistors (black), LEDs (blue), and the TCRT5000 sensor (purple).

To set up the IR sensor, the infrared reflected light is emitted by the sensor and reflected off the top of the demo car. These capture the reflected light implying analyzed the intensity, the sensor detects whether the car is present or not. Especially, a potentiometer as in figure 25 that

allows for adjusting the focus of the light. By using a small screwdriver to turn the potentiometer while monitoring the sensor output, the sensitivity to reflected light can be increased by turning it clockwise and decreased by turning it counterclockwise. The TCRT5000 sensor technology as in Figure 26 is beneficial for detecting parking space occupancy as it enables fast and accurate identification of vehicle presence.

The main purpose for choosing infrared (IR) sensors is their simplicity and ease of use. In other words, IR sensors are selected because of their uncomplicated nature and straightforward implementation. However, IR sensors can be affected by environmental sensitivity factors such as ambient light, temperature changes, and interference resulting in accuracy including the limit of detection requiring long-range and angle withholding the sensor.

2.1.2. ESP32

Figure 27 ESP32 Components

The ESP32 as in Figure 27, is a System-on-a-Chip (SoC) which is the main part of sensor module to integrated Wi-Fi and Bluetooth connectivity, along with a dual-core microprocessor. Allowing for seamless wireless communication and efficient processing power in a chip. Based on the Xtensa LX6 CPU architecture, which provides high-performance computing capabilities for executing complex tasks and handling processes simultaneously.

Including offers a wide range of built-in peripherals and interfaces, such as GPIO pins, UART, SPI, I2C, etc. These features enable easy integration

with sensors, actuators, and other components commonly used in IoT applications. The ESP32's efficient power management capabilities make for battery power. To allow devices to operate for extended periods without frequent battery replacements or recharging. Nevertheless, Understanding the intricacies of Wi-Fi and Bluetooth communication, as well as programming the ESP32, may require additional time and effort. Due to its extensive features, the ESP32 can be more complex to work with compared to simpler microcontrollers. Configuring and optimizing its functionalities may require a deeper understanding.

Despite the potential difficulty, the ESP32 versatility, integration capabilities, low power consumption, and strong development support make it a popular choice for IoT applications. Its ability to connect to Wi-Fi networks, communicate over Bluetooth, and continuously interact with various components of IoT projects enables reliable and efficient functionality.

Table 1 Comparison table of the microcontroller

	Arduino UNO R3	RasberryPi4	ESP32
WiFi built-in	No	Yes	Yes
Reasonable price	Yes	No	Yes
Power consumption	Medium	High	Low

After a thorough examination of table 1, it becomes evident that the ESP32 emerges as the premier choice for a sensor parking model, owing to its superior Wi-Fi capabilities, reasonable price, and commendable power efficiency. This assessment underscores the importance of selecting the ESP32 to achieve an optimal balance between functionality, affordability, and sustainability in the context of sensor parking systems.

2.2. Procedure

2.2.1. Circuit Diagram

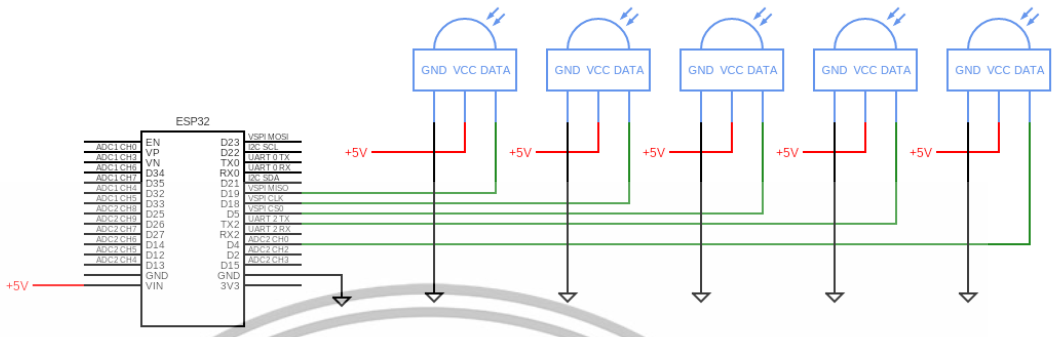


Figure 28 Circuit diagrams of sensor parking model

The circuit diagram as in figure 28 for the sensor model mock-up setup involves connecting multiple components, including five TCRT5000 sensors, an ESP32 development board, a breadboard, and jumper wires. Beginning to establish the power supply for the circuit by connecting the positive and negative leads of the power supply to the respective positive and negative rails on the breadboard. This ensures that all components receive power. Next place the TCRT5000 sensors on the module created. Each TCRT5000 sensor has a Digital (D0) port needed to connect with a digital pin on the ESP32 microcontroller. Use jumper wires to connect between each of the parts. The base pin of each TCRT5000 sensor is connected to specific pins on the ESP32 board. Connected Pin D4, TX2, D5, D18, and D19 of the ESP32 to the base pin of the 1,2,3,4, and 5 TCRT5000 sensor, respectively. That can be seen in Figure 28: Physical of IR sensor (TCRT5000). As an infrared emitter and phototransistor in a leaded package which blocks visible light to receive the reflection of the demo cars. These connections enabled communication and data received between the sensors and the microcontroller ESP32. To provide power to the TCRT5000 sensors, connect the VCC (power) pin of each sensor to the positive rail on the breadboard. Similarly, connect the GND (ground) pin of each sensor to the negative rail. This ensures that the sensors are properly powered and grounded within the circuit. By following these connections and setting up the circuit as described, you will establish the necessary electrical connections between

the TCRT5000 sensors and the ESP32 board, allowing for the detection and measurement of reflected infrared light.

2.2.2 Flowchart

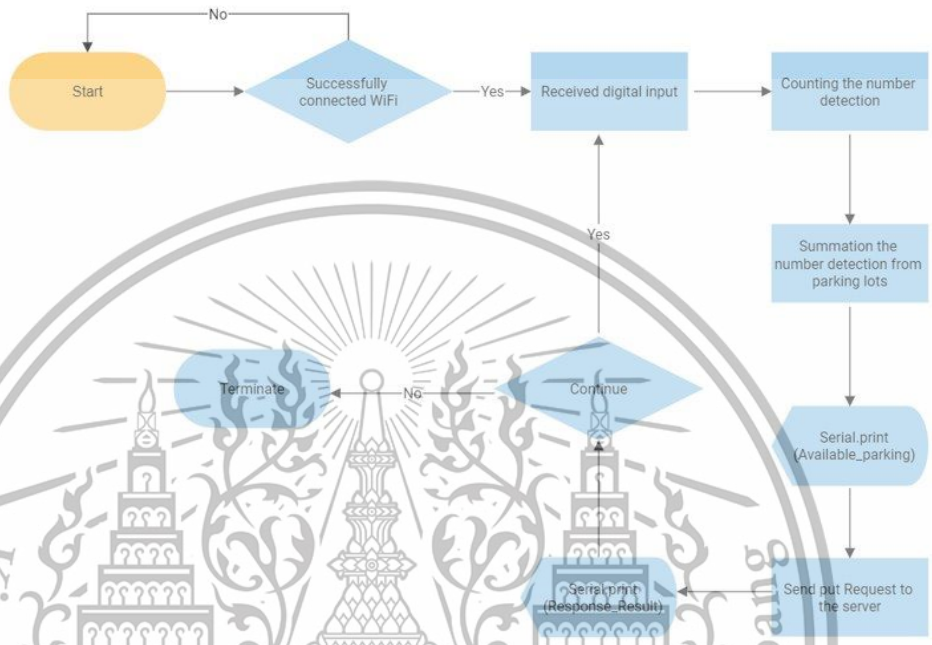


Figure 29 Flowchart diagram of sensor parking model

The program begins with checking whether the ESP32 is connected to a network, if the Wi-Fi connection is successful, the program proceeds to the next step. The infrared sensors (TCRT5000) are used to count the number of objects detected in the parking spaces. The counting detection number of the present cars are subtracted from the total number of parking spaces to determine the available parking spaces. The available parking is counted. After calculating the available parking spaces, the program sends an HTTP request to the server. The request includes the available parking count as a parameter. The program waits for a response from the server and then displays the result of the response. The program continues to loop, constantly monitoring the digital input from the IR sensors to detect any changes in the number of objects (demo cars) in the parking spaces. This loop ensures that the parking availability is regularly updated and accurate. The program will keep running in this loop until it is manually stopped, or

an end condition is specified in the code as in Figure 29. This allows for continuous monitoring and updating of the parking availability information.

3. IP camera scenario

3.1. Procedure

In the situation of outdoor parking, object detection is the better choice for the solution. We use TensorFlow pretrained model as a model for detecting and counting the number of parked cars, which is Faster RCNN ResNet50 [6].

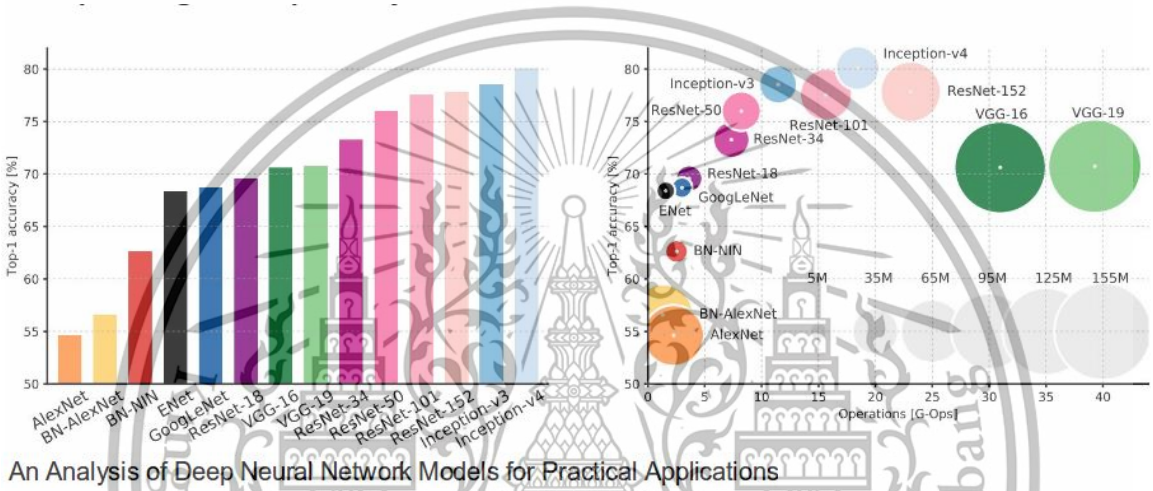


Figure 30: The deep learning obesity crisis. Lequertier, V. (2022, July 3).

According to Figure 30, the performance of ResNet50 nearly reaches the highest accurate CNN model. The reason for using Faster RCNN ResNet50 for detection is because this model is popular and accurate enough to detect only the cars in real-time. [12] [20]

First, after receiving image data with 15 fps and resolution size (1296,2304), then limit the object that will be detected by assigning the ID of the object. Since, the dataset of used pretrained model from COCO dataset, so the 'Car' class is 3.

There is a problem with the detection because it detects everything in the 'Car' class even though the confidence score is low. To avoid detecting low confidence score, the confidence score needs to be set, and during day and night, the ability of the detection is quite different.

During the nighttime, the detection is more accurate than during the daytime. To solve this problem, The input frames need to into grayscale.

Confidence Threshold

First, before converting the input frame into grayscale, since the confidence score during day and nighttime is quite different from each other, which makes it hard to determine the threshold. To ascertain the optimal value, the confidence threshold value begins with 0.25.

From the observation, with the confidence threshold value at 0.25, the accuracy of the detection during the daytime is lower than the detection during the nighttime.

The reason that the detection is better in the nighttime is because of the input colors, it has only black, white, and gray color. From this fact, we converted the input image frames into grayscale before passing through the process of detection [3].

After converting the input frames into grayscale, the confident percentage on grayscale at daytime is more than the normal one.

Figure 31 DAY detection without grayscale

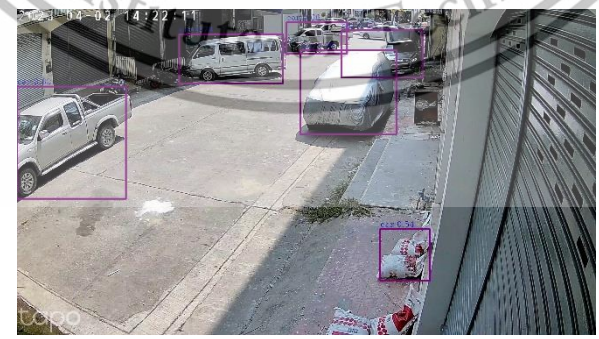


Figure 32 DAY detection with grayscale

Figure 33 NIGHT detection with grayscale

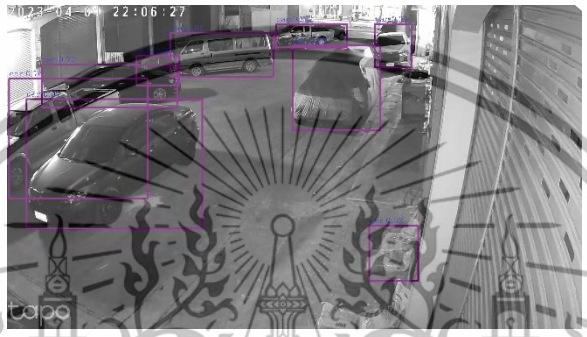


Figure 34 NIGHT detection without grayscale

Finally, the confidence threshold value can be determined at 0.5 is acceptable.

Grayscale

To increase the accuracy of daytime detection is to convert input VDO frame into grayscale by using OpenCV library, cv2.COLOR_BGR2GRAY and cv2.COLOR_GRAY2BGR, as shown in Figure 35.

```
#grayscale
frame_gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
frame_gray = cv2.cvtColor(frame_gray,cv2.COLOR_GRAY2BGR)
```

Figure 35 Convert VDO frame into grayscale.

The reason of using cv2.COLOR_GRAY2BGR is because use only cv2.COLOR_BGR2GRAY the color channel gives only 1 channel, so need to use cv2.COLOR_GRAY2BGR to make the color channel become 3 channels. TensorFlow technique needs 3 color channels of the input image data to do the process of detection.

For the detection, we use this grayscale frame as an input to do the detect process, but the result remains as the original VDO frame, other words mean that, detect at the grayscale frame, but output is display on the original frame.

Area and bounding box positions

Using only confidence thresholds and class determination are not enough to detect the car in specific area.

Figure 36 Region of Interest (ROI)

In figure 36, below the red line is an interesting area. To detect the cars only in the area under the red line is to determine the area of the bounding box and the position of the bottom right of the bounding box, the diagrams of the detect conditions are shown in figure 37 and figure 38.

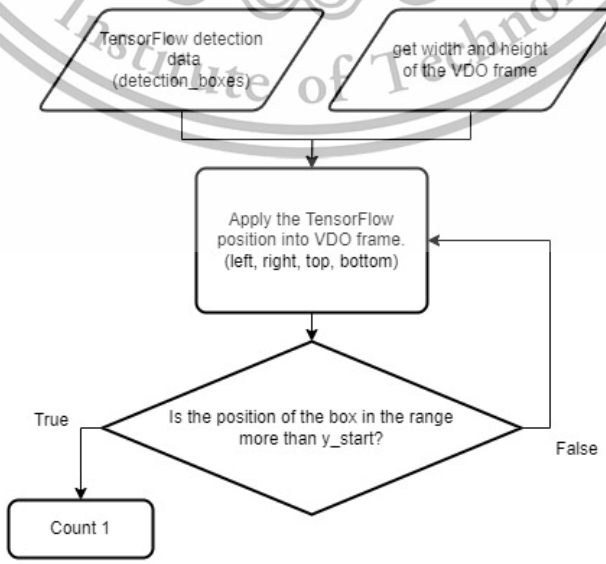


Figure 37 Area and Position of the bounding box condition

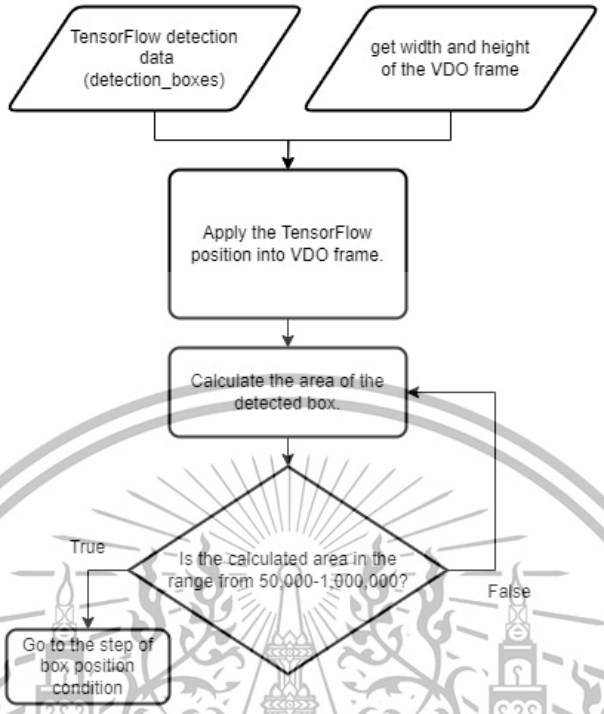


Figure 38 Area Calculation Conditions

As shown in figure 38, if the area of the bounding box is more than 50,000 but less than 1,000,000 is acceptable. If the box is small/too small/too large will not display and not count in the result.

To get the detection in only interested region is determined the position by using the bottom right position of the bounding box with the Y position of the frame.

Figure 39 Red line position

In figure 39, the position of 2 points (to create line, need 2 points) have a common Y position, which is 300. If the detection receives the position value of the bottom right of the box is more than 300, it is acceptable.

3.2 Flow Chart of detection process

Figure 40 Detection Process Flow Chart

4. Backend

4.1. Software

4.1.1. Falcon API

Falcon is a minimal, fast, and secure web framework for Python that allows you to build RESTful APIs with ease. It has no dependencies outside the standard library and works with both asyncio (ASGI) and gevent/meinheld (WSGI). The HTTP resources can be defined as classes and use class methods for different REST operations. [13]

There are many alternative tools for falcon API which each have own strength and weakness. Which is shown in table 2.

Table 2 Tools comparison

Framework	Performance	RESTful API Design	Minimalism	Speed of Development
Falcon	High (Built for speed and efficiency)	Excellent (Designed specifically for RESTful APIs)	High (Minimalist, bare-metal experience)	Moderate (Requires a bit more coding due to minimalism)
Flask	Moderate (Optimized for small to medium applications)	Good (Not specifically designed for RESTful APIs, but can be used to build them)	Moderate (Microframework but includes more built-in features than Falcon)	High (Simple and easy to use, rapid prototyping)
Django	Moderate (Can be slower due to its feature-rich nature)	Moderate (Needs Django REST Framework for RESTful APIs)	Low (Full-featured, providing many built-in functionalities)	High (Many built-in features speed up development time)
FastAPI	High (Designed for high performance)	Excellent (Designed specifically for RESTful APIs, supports OpenAPI standard)	Moderate (FastAPI includes more features than Falcon, but is still relatively minimalistic)	Very High (Python type hints, data validation, automatic documentation generation)

In comparison to from the Table 2, Flask, Django, and FastAPI, Falcon shines particularly in the areas of performance and RESTful API design. Falcon is built for speed and efficiency, making it an excellent choice for applications where performance is crucial. Its minimalist nature allows developers to have more control over their code and reduces the overhead associated with more feature-rich frameworks like Django. This minimalism might require a bit more coding, but it ensures that the Falcon application does exactly what you want it to do, and nothing more, resulting in clean, understandable code.

Furthermore, Falcon is designed specifically for creating RESTful APIs. This means it follows the principles and constraints of REST, providing a well-structured and easily maintainable API design. This focused design philosophy makes Falcon the go-to choose for HTTP API development.

While FastAPI also provides excellent support for RESTful API design and high performance, it does include more built-in features compared to Falcon. This could be viewed as an advantage, but it also means FastAPI carries slightly more overhead. If you prefer a bare-metal, low-level experience with no additional overhead, Falcon is the better choice.

In conclusion, for developers aiming to build highly efficient, scalable, and RESTful HTTP APIs, Falcon presents a compelling case. Its focus on minimalism, performance, and REST principles makes it a wise choice for any project where these attributes are valued.

4.1.2. MongoDB

MongoDB is a document database that allows the user to store and query data in a flexible and scalable way. It supports a distributed architecture with high availability, horizontal scaling, and geographic distribution. It also offers a unified query interface for various use cases, such as transactional, analytical, search, time series, and more. MongoDB is free to use. In database system, there are many software that can do similar job but it which have some advantages and disadvantages shown as table below. [14]

Table 3 Tools comparison

Database	Data Model	Scalability	Speed	Flexibility	Complexity
MongoDB	Document-Oriented	High (Horizontal Scaling)	High (Especially for Read Operations)	High (Schemaless)	Moderate (NoSQL)
MySQL	Relational	Moderate (Vertical Scaling)	Moderate (Optimized for complex queries)	Low (Requires predefined schemas)	High (SQL, ACID Transactions)
Cassandra	Wide-Column Store	High (Horizontal Scaling)	Moderate (Especially for Write Operations)	Moderate (Semi-Schemaless)	High (CAP Theorem, eventual consistency)
Redis	Key-Value Store	Moderate (Vertical Scaling)	Very High (In-memory Database)	Low (Simple Key-Value Data Model)	Low (Key-Value)

According to table 3 MongoDB, a document-oriented NoSQL database, is well-suited for handling Big Data and real-time applications. It offers high performance, especially for read operations and horizontal scalability, which allows it to handle increasing data or load by adding more servers to the database. This is a strength MongoDB holds over MySQL, which traditionally scales vertically by adding more power to a single server.

Compared to Cassandra, MongoDB offers a more flexible data model. It stores data in a BSON format, which is a binary representation of JSON-like documents. This document model makes MongoDB extremely versatile, as it can store complex data types and structures, and it doesn't require a predefined schema, unlike MySQL.

While Redis has a speed advantage due to its in-memory nature, it's primarily a key-value store with less flexibility and complexity in data

structuring compared to MongoDB. Redis is ideal for caching and message queuing, but if your application requires complex data manipulation and analysis, MongoDB is a more suitable choice.

In conclusion, since the application requires a flexible, scalable, and high-performance database that can handle complex data and structures, MongoDB is a compelling choice. Its features make it ideal for a variety of applications, including content management, mobile and social infrastructure, real-time analytics, and many more. Its design principles provide developers with the tools to handle growing data demands of the modern web and the emerging Internet of Things (IoT) applications.

4.1.3. Nginx

Nginx is a software that can perform various web tasks, such as serving web pages, proxying requests, caching content, load balancing traffic, and streaming media1. It was created by Igor Sysoev to solve the problem of handling many concurrent connections efficiently. It is open-source and widely used by many popular websites. It has a modular architecture and supports many protocols and standards. There are three main popular software that are currently used which are Nginx, Apache and ISS, each strength and weakness are described in the table below. [15]

Table 4 Tools comparison

Web Server	Performance	Scalability	Flexibility	Configuration	Security
Nginx	High (Event-driven model)	High (Supports many concurrent connections)	High (Also can be used as a load balancer or HTTP cache)	Moderate (Directives structure)	High (With ModSecurity)
Apache	Moderate (Process-based model)	Moderate (Can struggle with many concurrent	High (Rich feature set via modules)	Moderate (Directive based, .htaccess support)	High (With ModSecurity)

		connections)			
IIS	Moderate (Thread-based model)	Moderate (Requires more resources for many connections)	Moderate (Integrated with Windows features)	High (GUI and command line)	High (Built-in Windows authentication)
LiteSpeed	High (Event-driven model)	High (Supports many concurrent connections)	Moderate (Main focus is on speed)	Moderate (Similar to Apache)	High (Built-in anti-DDoS features)

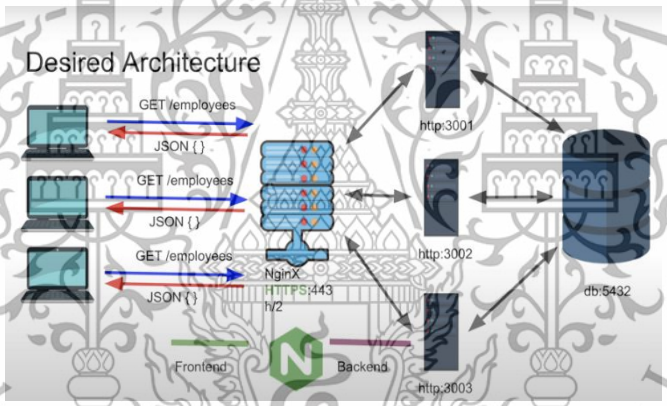


Figure 41 Demonstration of Nginx position in network

Nginx, with its event-driven architecture, can handle a large number of concurrent connections and is known for its high performance, outperforming Apache and IIS, which follow a process-based and thread-based model, respectively. This architecture allows Nginx to efficiently serve static content and handle high traffic situations.

While Apache and IIS offer a broad range of features and flexibility, they can struggle when dealing with many simultaneous connections, unlike Nginx and LiteSpeed. Furthermore, IIS is mostly limited to the Windows environment, which can be a limitation if your infrastructure includes Linux servers.

LiteSpeed also employs an event-driven model, similar to Nginx, and is known for its speed and efficiency. However, Nginx offers more flexibility, allowing it to be used not only as a web server but also as a reverse proxy, mail proxy, load balancer, and HTTP cache.

In terms of configuration, Nginx follows a directive structure, which might require some learning for newcomers but offers a robust and flexible configuration. Security-wise, Nginx can integrate with ModSecurity, a popular open-source web application firewall, ensuring robust protection for your applications.

In conclusion, Nginx's ability to handle a high number of concurrent connections, its flexibility to be used in multiple roles, and its robust security make it an excellent choice for modern web applications. Whether you're serving static content, dynamic pages, or operating as a reverse proxy, Nginx provides a highly scalable, secure, and efficient solution.

4.1.4 Gunicorn

Gunicorn is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model ported from Ruby's Unicorn project. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and speedy.

Table 5 WSGI Server comparison

WSGI Server	Configuration	Performance	Compatibility	Asynchronous Work	Language Support
Gunicorn	Easy (Simple command line arguments)	High (Efficient pre-fork worker model)	High (Pure-Python implementation, WSGI compliant)	Partial (through gevent or eventlet workers)	Python
uWSGI	Moderate (Extensive options can be overwhelming)	High (Efficient process/spooler model)	High (WSGI, PSGI, Rack compliant)	Yes (Built-in functionality)	Multi-language (Python, Ruby, Perl, etc.)

Apache	High (Requires Apache configuration knowledge)	High (Integrated with Apache's process model)	High (WSGI compliant)	No	Python
Daphne	Easy (Integrated with Django Channels)	Moderate (ASGI server, not optimized for WSGI apps)	Moderate (ASGI server, compatible with Django Channels)	Yes (Built for asynchronous apps)	Python

4.2. OS

4.2.1. XCP-NG

XCP-NG OS is a Linux-based operating system that allows you to run multiple virtual machines on a single server. It is derived from XenServer, an open-source project that provides a high-performance and secure virtualization platform. XCP-NG OS is developed by a community of users and developers who aim to offer unrestricted features and open-source accessibility. It is categorized as a Type-1 Hypervisor.

4.3. Procedure

4.3.1. Preparation

Before setting up the backend, ensure that all necessary software and the operating system are installed. This includes the Falcon API framework, MongoDB, Nginx, and the XCP-NG system. If any of these components are not installed, follow the respective official documentation to complete the installation. Refer to APPENDIX C.

4.3.2. Setup Database

Begin by initializing a MongoDB database. MongoDB, as a NoSQL database system, is excellent for managing large amounts of unstructured

data. In this case, you will need a collection called 'park' within the database to store the details of parking areas.

Here's the structure for each document in the 'park' collection:

_id: A unique identifier generated automatically by MongoDB.

id: A unique identifier specific to each parking area.

name: The name of the parking area.

location: An embedded document containing latitude (lat) and longitude (lng) coordinates of the parking area.

details: An array of embedded documents. Each of these represents a specific part of the parking area, containing its name, the current number of parked cars (current), and the total parking capacity (total).

contacts: An array of embedded documents with details of each contact associated with the parking area. This includes the contact's name, their logo (represented as a URL), the bot name, and the endpoint URL for the bot.

The 'park' collection will look like this in MongoDB database:

Figure 42 MongoDB Park Collection Document Structure

4.3.3. Create API using Falcon

The Falcon framework is a minimalist WSGI library used for building robust and efficient cloud APIs and app backends. Using Falcon, you'll develop RESTful API endpoints for the various operations that need to be performed.

This is a brief outline of the API endpoints required in the application

ParkUpdate Endpoint: This endpoint allows the current status of a particular parking area to be updated. Here is an example of how you might define the ParkUpdate class and its methods:

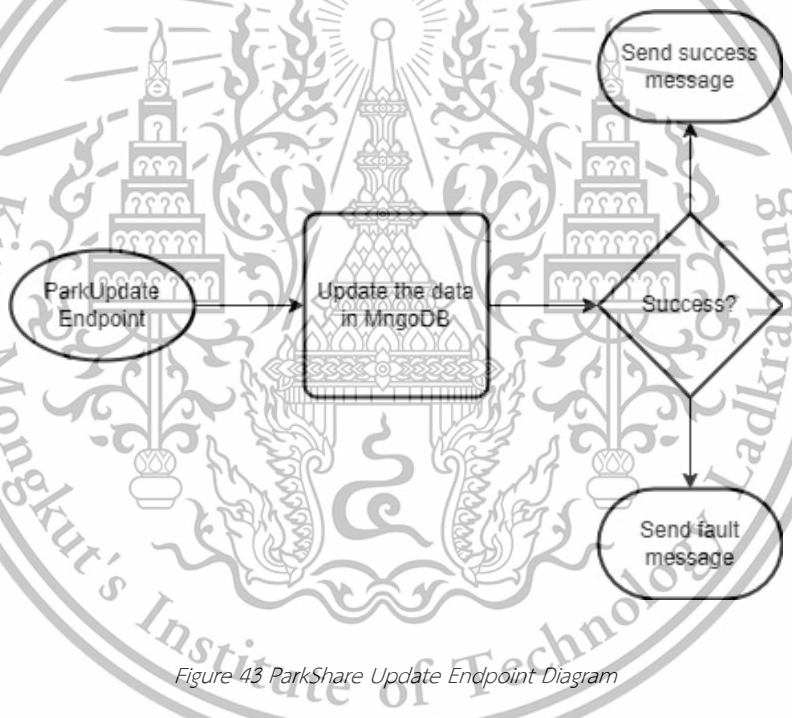


Figure 43 ParkShare Update Endpoint Diagram

With this, a PUT request to /park/update would trigger an update to the relevant document in MongoDB collection.

ParkCreate Endpoint: This endpoint allows the creation of new parking areas. Here's how to define the ParkCreate class:

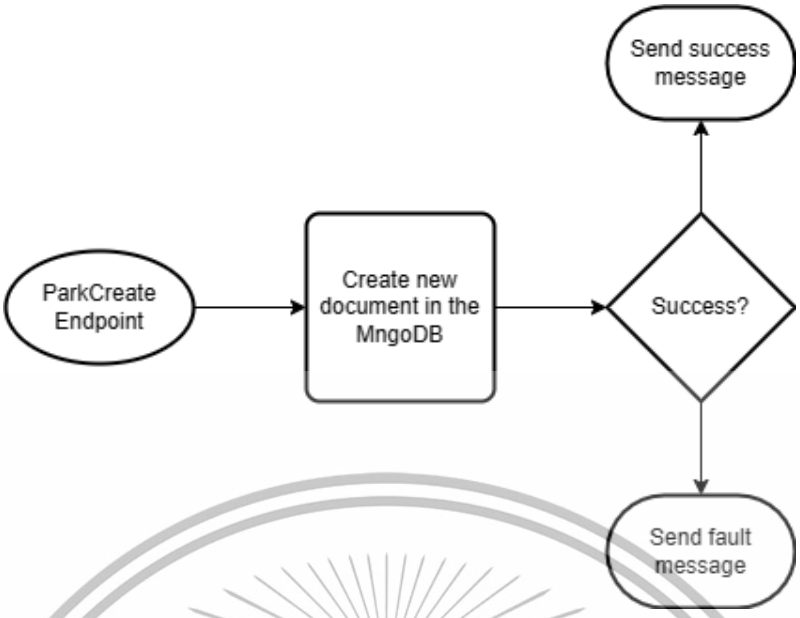


Figure 44 ParkShare Create Endpoint Diagram

With this, a POST request to /park/create would create a new document in MongoDB collection.

ParkGet Endpoint: This endpoint retrieves the data for a specific parking area. Here's an example of the ParkGet class:

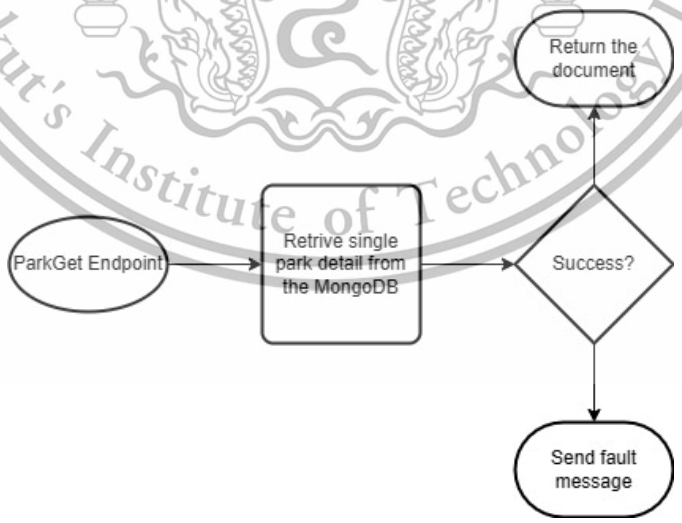


Figure 45 Park Get Endpoint Diagram

With this, a GET request to /park/get with a specific id would return the details of the corresponding parking area.

ParksGet Endpoint: This endpoint retrieves data for all parking areas. Here's an example of the ParksGet class:

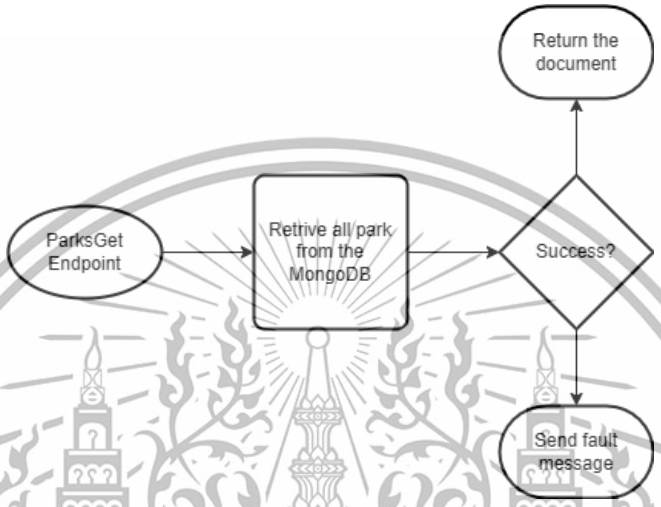


Figure 46 ParkGet Endpoint Diagram

With this, a GET request to /parks/get would return the details of all parking areas.

4.3.4. Integration of Sensor and Camera Data

To effectively manage and allocate parking spaces, the application needs to collect real-time data. This data collection is achieved by integrating sensor and camera data.

Sensor Data: Sensors placed in each parking space detect whether a car is parked in the spot or not. This could involve infrared sensors, ultrasonic sensors, or any other suitable sensing technology. For instance, the TCRT5000 infrared sensor can be used to detect the presence of a vehicle. These sensors send their data to the ESP32 microcontroller, which processes the data and sends updates to the backend server.

Camera Data: IP cameras installed in the parking lot capture live images of the area. These images can be processed using image processing algorithms and techniques such as AI-based object detection and recognition to determine the occupancy of the parking spaces.

The AI image detection can employ Convolutional Neural Networks (CNN), Region-based Convolutional Neural Networks (RCNN), Fast RCNN, Faster RCNN, or ResNet50 for efficient and accurate object detection. These AI models can be trained to recognize cars and other relevant objects in the parking area.

Data Integration: The sensor and camera data need to be integrated for the application to provide a comprehensive view of the parking area's status. This is achieved by collecting data from the sensors and cameras, processing it, and storing it in the MongoDB database.

The Park Update API endpoint can be used to update the current status of the parking area based on the collected sensor and camera data. The update involves a PUT request to **/park/update** with the relevant data. The backend then processes this request, updates the MongoDB database accordingly, and returns a success or error message.

By effectively integrating sensor and camera data, the application can provide accurate real-time information about the availability of parking spaces to users.

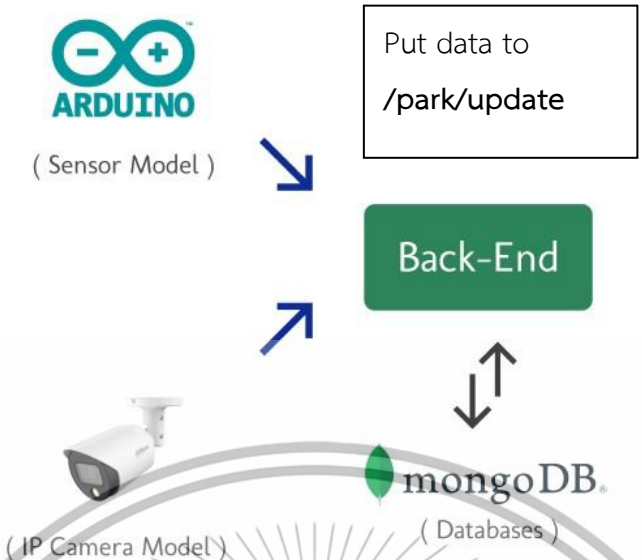


Figure 47 Backend workflow

4.3.5. Set up Nginx

Nginx acts as a reverse proxy that routes the traffic to our Falcon API and manages the HTTP(S) requests. Configure Nginx to direct incoming requests to the appropriate Falcon API endpoints. Generate and set up SSL certificates to enable secure communication.

4.3.6. Testing

Conduct comprehensive testing of all the components and the entire system. This includes functionality testing of the API endpoints, performance testing under varied loads, security testing to spot any potential vulnerabilities, and testing the sensor and camera data integration process to ensure accurate data updating in the app. The application has been testing as follow.

a. Functionality testing

In this test the function of the application has been test to ensure that the application are working correctly. The postman has use for testing the endpoint due to it ease of use. The result are as shown as in Figure 48.

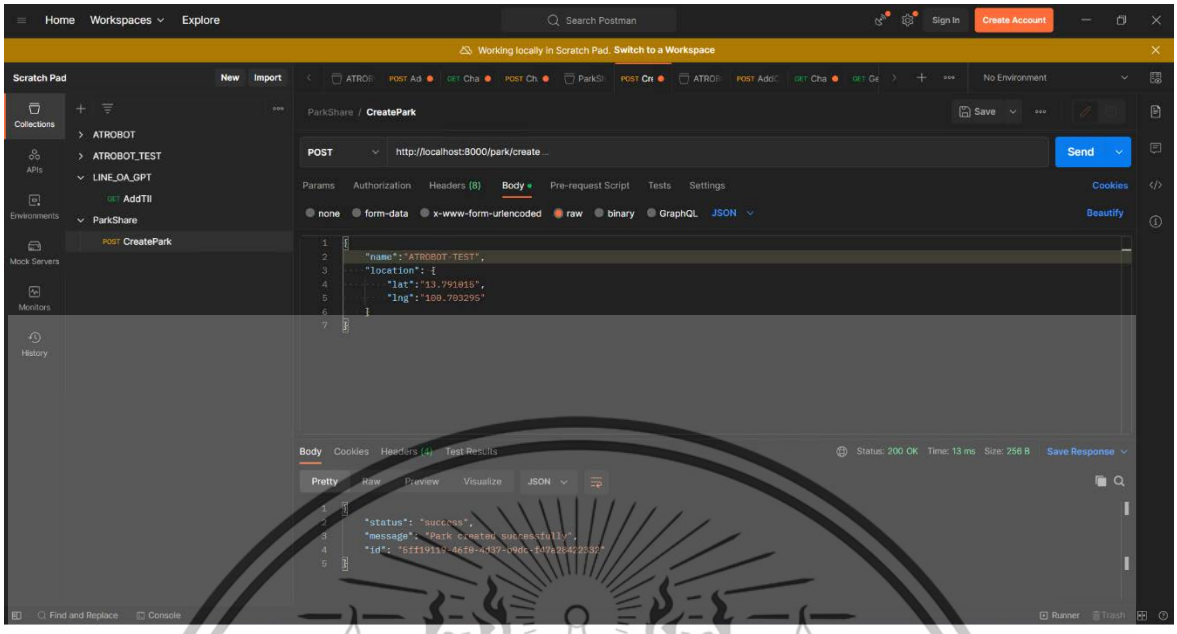


Figure 48 Test create new location (/park/create) endpoint using Postman.

Table 6 Test result of each endpoint

Endpoint	Pass	Fail
/park/update	✓	
/park/create	✓	
/park/get	✓	
/parks/get	✓	

b. Performance testing

Performance testing is a key element in software engineering that focuses on determining the responsiveness, reliability, and stability of a system under different workload conditions. A tool known as Apache JMeter has been utilized to evaluate the server's performance under different circumstances, such as multiuser concurrent usage.

The testing conditions are as follows

Concurrent user : 100 users

Ramp up time : 1 seconds

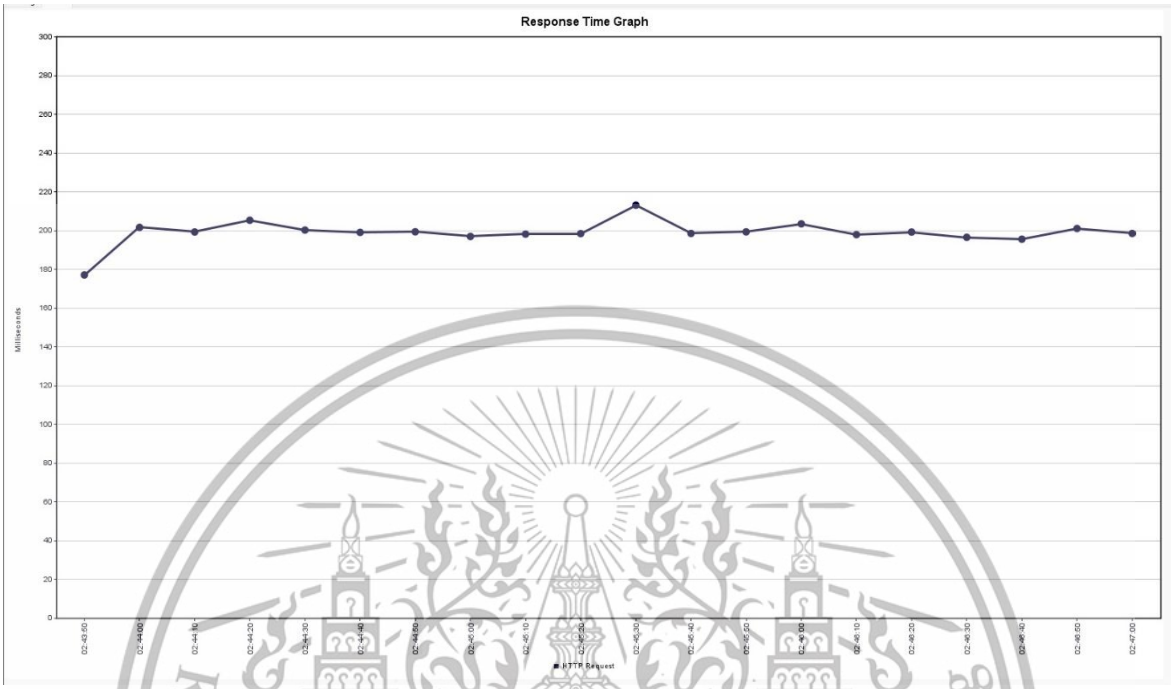


Figure 49 Response time of the server

As the Figure 49 shown the response time of the server have increased a little bit after the ramp up process but the over all server response are below 200 milliseconds which are good.

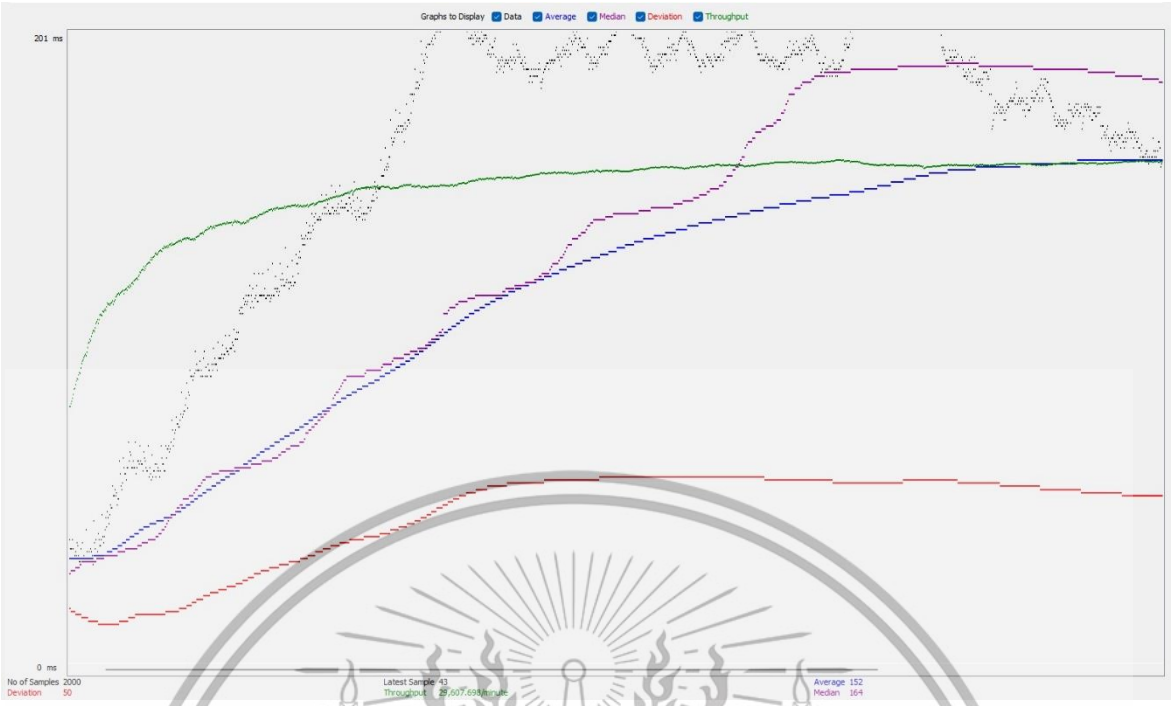


Figure 50 Graph of server performance in detail

In the Figure 50. This graph show more the detail in depth. The variable name are as follows.

Throughput (Green line): This represents the ability of a server to handle heavy loads. It measures the number of requests processed per unit of time (seconds, minutes, hours) by the server. A higher value indicates better performance of the application.

Deviation (Red line): Also referred to as Standard Deviation, it is a measure of the amount of variation or dispersion of a set of values. A low standard deviation indicates that the values tend to be close to the mean (average) of the set, while a high standard deviation indicates that the values are spread out over a wider range.

Median (Purple line): This is the middle point of a data set; 50% of the samples are below this point, and 50% are above it. In other words, it's the 50th percentile. Median is a better measure than the average when a data set is skewed (has outliers).

Average (Blue line): This is the sum of all measured values divided by the number of measurements. It provides a general idea of the response time of your server or application. However, the average can be skewed if there are data outliers.

Data (Gray line): In the context of a JMeter graph report, this refers to the various metrics and statistics generated from the test results, such as response time, latency, throughput, number of successful requests, etc. The data is used to plot the high throughput indicates that there are many user talking with the backend resulting in high data traffic. And the data that has been send and receive are increasing according to the server. The inconsistency ramps up of the data due to the number of thread that the server have been use. While all thread are responding to the user the request have to wait a little longer and the server will spawn more worker and thread to handle more traffic.

In conclusion, the graph show that server and the backend is capable of serving 100 user concurrent which is excellence for real life usage.

4.3.7. Deployment

The deployment process is the final stage of bringing park sharing applications to life, making it available for users to interact with. Here are the steps to might follow

Virtual Machines Deployment with XCP-NG: Next, the application is deployed on a virtual machine. For this, we are using XCP-NG, an open-source virtualization platform. You create a new virtual machine on your XCP-NG host, configure it according to the requirements of your application (like CPU, memory, and storage), and then deploy your Code on this VM. You can then start your code using Gunicorn and your application will be up and running.

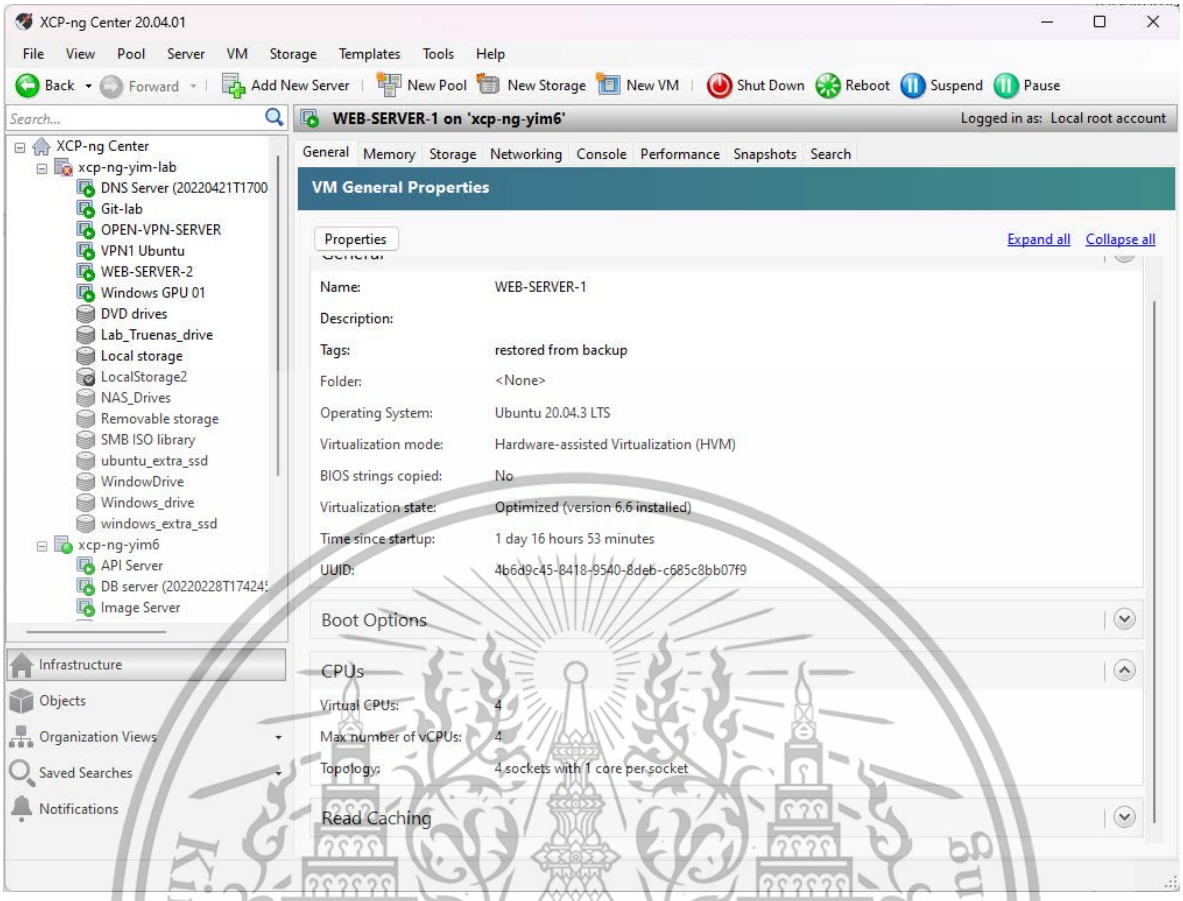


Figure 51 Virtual machine specification

Nginx as Reverse Proxy: As a part of the deployment process, also set up Nginx, a popular open-source web server and reverse proxy server. Nginx can efficiently handle and distribute incoming network traffic to your application, enhancing its performance, security, and scalability. In the context of application, Nginx is configured to pass incoming HTTP requests to your Falcon server.

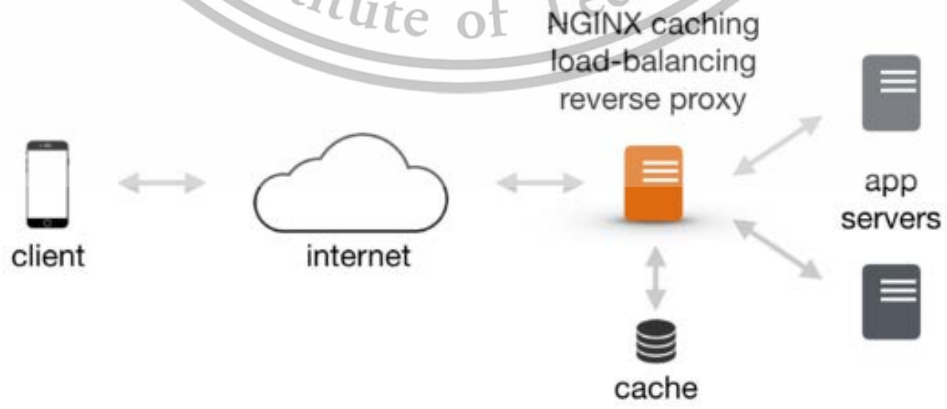


Figure 52 Nginx reverse proxy

Testing and Verification: After the application is deployed, it is necessary to conduct comprehensive testing to ensure that all the components are working as expected. Test all API endpoints, check the connectivity with the MongoDB database, and ensure the application accurately reflects the status of the parking areas based on the sensor and camera data.

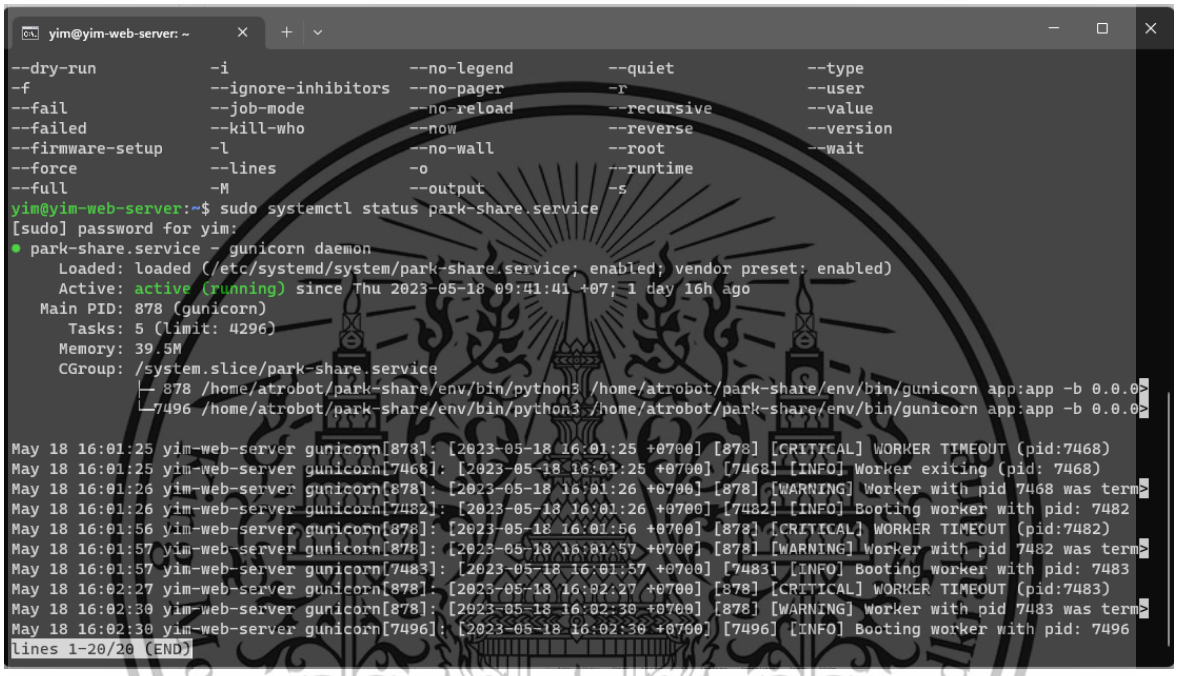


Figure 53 Test and check if software is running.

4.3.8. Maintenance and Updates

Post-deployment, it is important to continually monitor the system performance and carry out regular maintenance. Regular database backups, software updates to Falcon, Nginx, and XCP-NG, and periodic security reviews should be part of this process. This involves several steps:

Regular Updates: Regular updates to the software components of the system, including the Falcon framework, the database (MongoDB), and the virtual machine software (XCP-NG), are important. These updates often contain security patches and feature enhancements that can improve the performance and security of your application.

Continuous Monitoring: Continuous monitoring of the application can help identify and address any issues or bugs in real-time. This could involve monitoring the system's performance, the database for any inconsistencies, and user feedback for any problems they might be experiencing. This will help to ensure the smooth operation of the application and a good user experience.

Regular Backups: Regular backups of the application's data are important for data security and integrity. It is recommended to have two types of backups:

Snapshot Backup: This is a backup of the system at a particular point in time, creating a "snapshot" of the data. These backups should be performed every 3 days. Snapshot backups are relatively quick and do not take up a lot of storage space. They can be used to restore the system to a specific point in time if necessary.

Full Back up: This is a complete backup of all data in the system. Full backups should be performed every 7 days. This type of backup takes more time and storage space compared to snapshot backups, but it provides a complete copy of all data. In the event of a system failure or data loss, a full backup can be used to restore the entire system.

Bug Fixes and Feature Enhancements: Based on feedback from users and monitoring of the application's performance, there may be a need to fix bugs or add new features to the application. These changes should be implemented in a controlled manner, with thorough testing before they are deployed to the live application.

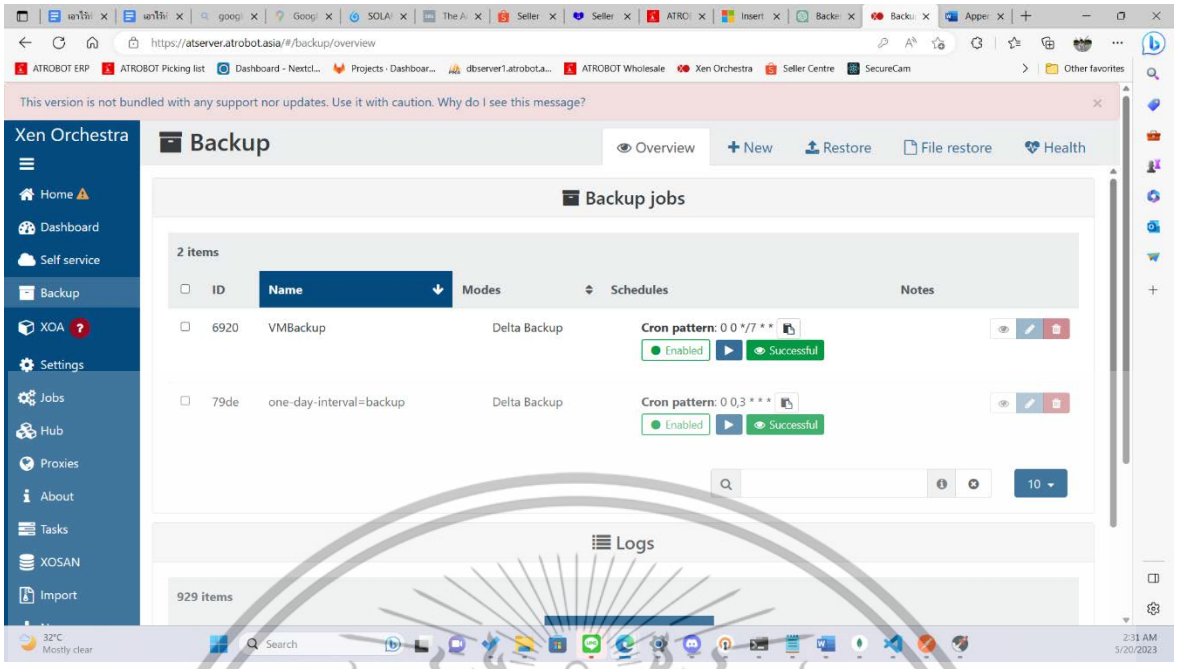


Figure 54 Automatic backup process

This procedure creates a robust, secure, and efficient backend for the park sharing application. The design is modular, allowing for future enhancements and expansions as required.

5. Frontend

5.1. Software

5.1.1. React JS

React JS is a library that helps you create user interfaces using components. Components are pieces of code that define how a part of the UI should look and behave. React JS is not a framework, and it can work with other libraries to target different platforms, such as the web or mobile. React JS uses a syntax called JSX, which is a mix of JavaScript and XML, to write components that tell React JS what to display on the screen. React JS also handles much of the rendering logic for you, letting you concentrate on the UI design]

Table 7 Framework comparison

Framework/Library	Learning Curve	Performance	Community & Ecosystem	Flexibility	Use Case
React	Moderate (JSX syntax, uni-directional data flow)	High (Virtual DOM)	Large (Created by Facebook, large community and wide range of libraries)	High (Very flexible, mix-and-match architecture)	Single-page applications, mobile applications (with React Native), complex UIs
Angular	High (TypeScript, complex syntax, decorators)	Moderate (Real DOM)	Large (Created by Google, mature)	Low (Opinionated, full-featured framework)	Large, enterprise-level applications
Vue	Low (Simple syntax, easy learning curve)	High (Virtual DOM)	Moderate (Increasingly popular, growing community and ecosystem)	High (Mix-and-match architecture, but also provides official supporting libraries)	Single-page applications, smaller projects, rapid prototyping
Svelte	Low (Compiles to vanilla JS, simple syntax)	Low (Compiles to vanilla JS, simple syntax)	Small (Newer, smaller community and ecosystem)	Moderate (Reactive programming model)	Single-page applications, less complex UIs

React stands out with its high performance, thanks to its Virtual DOM which minimizes costly DOM operations. React also uses a component-based architecture which allows for reusability and better organization of code. This can lead to increased developer productivity and easier maintenance compared to Angular complex syntax and decorators.

Although Vue has a lower learning curve and a similar architecture to React, React community and ecosystem are much larger. This means that you're more likely to find pre-built components or libraries for specific requirements in React, which can greatly speed up the development process.

Svelte offers excellent performance due to its unique approach of shifting work from the browser to the compile step. However, its community and ecosystem are smaller than React, which means fewer resources, libraries, and tools for developers to use.

React is also highly flexible, offering a "mix-and-match" approach that allows you to choose only the pieces of the library that you need, and to integrate with other libraries as necessary. This stands in contrast with Angular opinionated, full-featured approach, which includes a lot of features that may not be needed and can add unnecessary complexity.

React is also suitable for building mobile applications through React Native, which shares a lot of its syntax and concepts with React. This makes React a versatile choice that can handle a wide range of use cases, from web to mobile.

In conclusion, React's performance, large community and ecosystem, flexibility, and broad use case applicability make it a compelling choice for building fast, interactive user interfaces for a wide range of applications.

5.2. Procedure

The procedure for the front-end development of the park sharing application is based on a well-thought-out strategy, designed to meet the requirements of the end users. It involves the careful selection of appropriate software technologies, UI/UX design, and the integration of various services, data, and interfaces.

The frontend development process can be broken down into the following steps:

5.2.1. Planning and Requirement Analysis

In the initial phase, the primary objective is to gain a comprehensive understanding of the project's specific requirements. For the park sharing application, the primary functionality centers around two crucial elements: displaying location and parking availability and presenting map visualization.

Displaying Location and Parking Availability: The application should provide an interface that accurately displays each parking spot and its availability status in real time. This feature will require the frontend to seamlessly communicate with the backend, which will provide the necessary data through APIs. The front-end should be designed to promptly and effectively present these updates to the user.

Map Visualization: The application should include an interactive map feature, allowing users to visually identify the location of parking spots. The map should provide options for zooming and panning to provide users with a comprehensive view of various locations. Integration with a mapping service, such as Google Maps or Mapbox, will be necessary to implement this feature. The UI should be designed to seamlessly incorporate this map service and present it intuitively to the user.

With these requirements clearly defined, the design and development stages can progress with a specific target outcome, ensuring that all crucial features are effectively implemented into the final application. This meticulous planning phase is key to guaranteeing a seamless user experience, while simultaneously meeting the core functional requirements of the project.

5.2.2. Selection of Frontend Technologies

The choice of front-end technologies for this project is pivotal, given the requirements and complexity of the task at hand. Considering the following technologies have been chosen:

React.js: The main technology for building the application's user interface is React.js. This JavaScript library is renowned for its efficiency and flexibility in building interactive UIs. Its component-based structure encourages reusability and offers ease in managing the state of components, an aspect crucial for real-time updates such as changes in parking availability.

React Libraries: To enhance the functionality and efficiency of the project, several React libraries will be utilized. These include:

React-Router: This library will facilitate routing in the application. It allows for the creation of distinct routes that lead to different pages, thereby enabling the rendering of different components based on the route chosen.

OpenLayers: For the task of map visualization, Open Layers has been chosen. This high-performance, feature-packed library allows for easy creation of interactive maps. Its compatibility with React will enable seamless integration into the application to provide users with an intuitive and responsive map interface.

HTML & CSS: HTML and CSS will be employed for the basic structuring and styling of the application. In a React.js environment, HTML is used within JavaScript code (as JSX), and CSS can be applied inline or modularly based on the design requirements of the project.

Through careful selection of these technologies, the team is well-equipped to create a robust and user-friendly park sharing application that accurately meets the outlined requirements.

5.2.3. Designing the User Interface

With requirements and technologies in hand, the process of designing the User Interface (UI) begins. This includes the creation of a layout, selecting color schemes, typography, button styles, and other visual elements. Tools like Adobe XD, Sketch or Figma may be used to create a mockup of the application's interface.

5.2.4. Developing and Integrating Components

This phase involves converting the UI/UX designs into functional code using React.js and the selected libraries. The application is divided into several components, each handling a distinct part of the application. The primary components that need to be developed are

Left Panel Component: The Left Panel component is designed to display details about the selected parking spot, such as its name, availability status, and other relevant details. This component is hidden by default and becomes visible when a user either clicks on a parking location marker on the map or searches for a specific location. This dynamic display ensures a cleaner UI while providing the necessary information on demand.

Map View Component: The central part of the screen is dedicated to the Map View component. This component, created using the Open Layers library, shows an interactive map with markers representing parking locations. Users can zoom in, zoom out, and pan across the map to view different areas. Selecting a marker activates the Left Panel component, displaying detailed information about the chosen parking spot.

Right Panel Component: On the right side of the screen is the Right Panel component. This section serves as a communication interface for users to interact directly with the service of each location. This could be in the form of a chatbot or a direct message interface that allows users to ask questions or receive updates about the parking location.

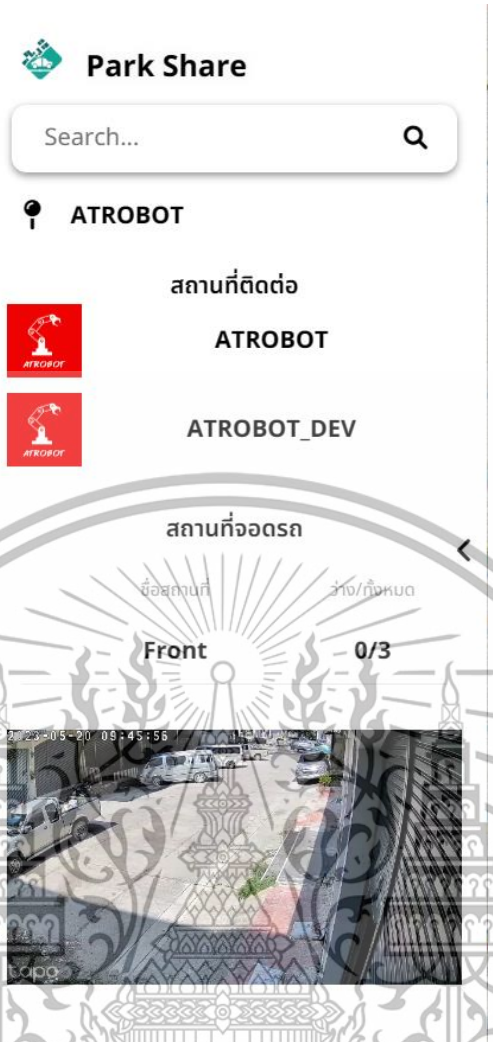


Figure 55 Left Panel Image component

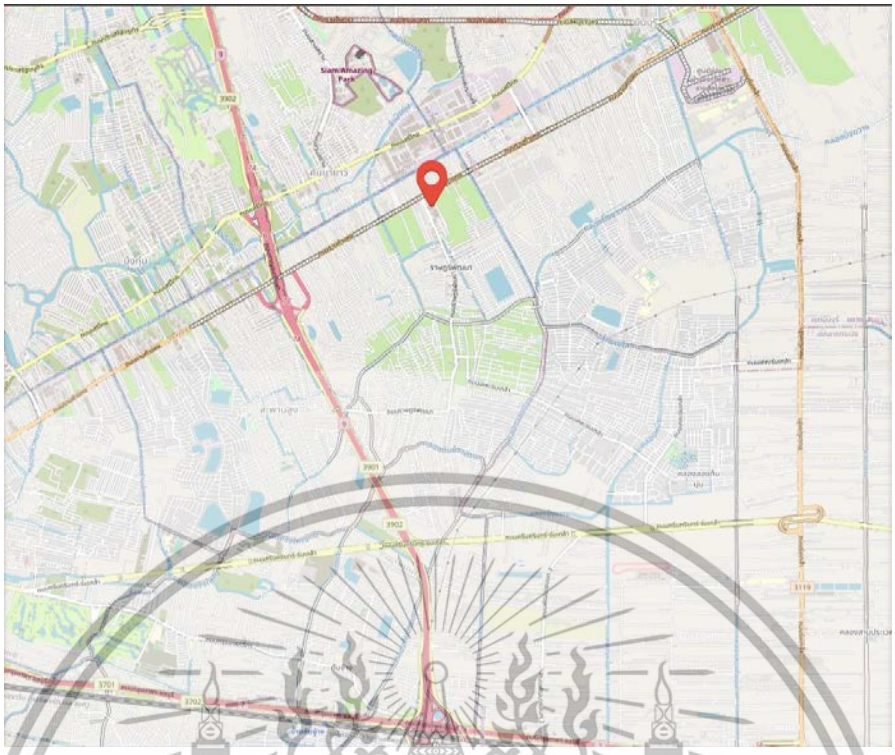


Figure 56 Map view component

Figure 57 Right panel component

These components are developed individually and then integrated to form the complete frontend of the application. Each component communicates with the backend via APIs to fetch and display real-time data, such as parking availability. This ensures that users have access to the most up-to-date information.

In essence, the development and integration of components is a critical process that transforms static design mockups into a fully interactive, dynamic application.

5.2.5. Testing

The testing phase is integral to the development process. It ensures the application's functionality, usability, performance, and compatibility meet the desired standards. The tests should cover every aspect of the application, from individual components to the overall user experience. Below are the types of testing performed:

Unit Testing: This is the first level of testing, where each individual component of the application is tested. The React application will have many components, and each one will have its specific functionality. Using tools like Jest, each component's functionality and rendering can be tested to ensure they work as expected.

Integration Testing: After unit testing, the components are integrated, and testing is performed to check if they work together correctly. This will involve testing the APIs and ensuring that data is correctly fetched and displayed on the front end. Tools like Enzyme can be used alongside Jest for these tests.

System Testing: This phase of testing involves testing the software as a whole to validate that it meets the specified requirements. All the components are combined, and the entire application is tested as a complete system to ensure that it behaves as expected in different scenarios.

Usability Testing: Usability testing involves checking the application's user interface and user experience. It verifies that the UI is intuitive and easy to understand, that the application is user-friendly, and that users can use it without encountering issues or confusion.

Based on this phase of testing. The first version of UX/UI has changed due to it user-friendly, and some functionalities have been changed from show parked/total to free/total to ensure the best user experience.

Performance Testing: This tests the software under various loads to determine how it behaves under stress and if it can handle many users at once. It is important to conduct performance testing to ensure that the application can maintain its speed and efficiency even when multiple users are using it simultaneously.

Compatibility Testing: The application should be tested on various browsers (Chrome, Firefox, Safari, etc.), devices (desktops, tablets, mobiles), and screen sizes to ensure its compatibility across different platforms and environments.

5.2.6. Deployment

The deployment phase is the final stage in the development cycle where the application is made available for end-users. For this project, a private server is utilized for hosting the application. The steps involved in the deployment process are as follows:

Server Setup: The private server should be appropriately configured to run the React application and the Nginx server. This includes installing necessary software like Nginx.

Application Build: On the development machine, run the build command (npm run build or yarn build depending on the package manager used). This command will create an optimized production build of the React

application. The output will be a build directory with static files that make up the application.

File Transfer: Once the build is successful, the static files need to be transferred to the private server. This can be done using various file transfer methods such as FTP, SCP, or rsync.

Nginx Configuration: On the server, Nginx needs to be configured to serve the static files of the application. This involves modifying the Nginx configuration file (usually located at /etc/nginx/sites-available/default in Linux-based systems). The root of the server block in this file should be pointed towards the directory containing the transferred build files. After making changes, it is important to check the configuration for any syntax errors and then restart or reload Nginx to apply the changes.

Firewall Settings: Ensure that appropriate firewall settings are in place to allow traffic to the Nginx server. This typically involves allowing inbound connections on port 80 (HTTP) and port 443 (HTTPS if SSL is configured).

Testing: After the deployment, carry out a final round of testing to ensure that the application is running as expected. This includes checking all features of the application and making sure it loads correctly with no console errors.

By following these steps, the React application will be successfully deployed to the private server and accessible to end-users. It is recommended to monitor the application after deployment for any unexpected issues or errors and to set up a continuous integration/continuous deployment (CI/CD) pipeline for efficient handling of future updates and improvements.

5.2.7. Feedback and Iteration

After the application is deployed and is in use, the next phase involves gathering feedback from users and iterating on the application based on that feedback. This is a crucial step in the lifecycle of an application, as it allows for continuous improvement and ensures the application stays relevant and useful to its users. The process can be broken down into the following steps:

Feedback Collection: Users of the application are the best source of feedback. Their interaction with the application in real-world scenarios can provide insights that are not easily available from the testing environment. Feedback can be collected via surveys, user interviews, suggestion boxes, or direct communication channels in the application. Feedback should be encouraged on all aspects of the application, such as usability, performance, and feature suggestions.

Feedback Analysis: Once the feedback is collected, it needs to be thoroughly analyzed. This analysis can identify common issues faced by users, potential improvements, and new features that can be added. It is essential to prioritize feedback based on the severity of issues and the potential impact of improvements.

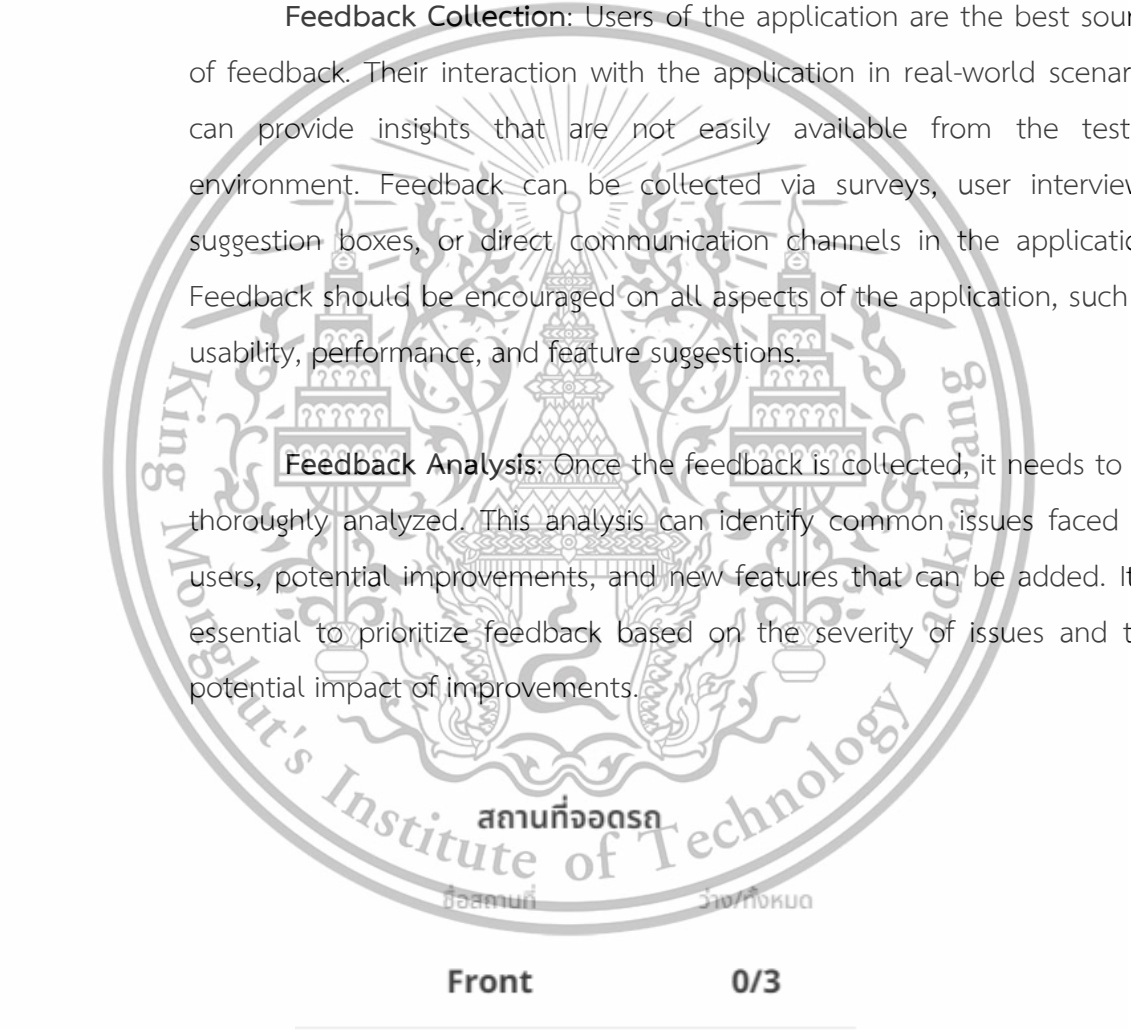


Figure 58 Park available status

Based on user feedback the previous were current/total which can cause confusion on some user. The display has been changed to be free/total for better experience.

Planning Iterations: Based on the feedback analysis, the next iterations of the application are planned. This involves deciding on the changes to be made, the features to be added, and any performance improvements that can be implemented. These changes should be planned and organized into manageable tasks.

Implementing Changes: The development team then works on implementing the planned changes. This involves coding the new features, making necessary modifications, and testing these changes thoroughly before they are released. This step is essentially a mini version of the original development cycle.

Deploying Iterations: Once the changes have been tested and verified, they can be deployed to the production environment. It is recommended to inform users about the updates and changes that have been made.

Repeat the Process: This entire process is cyclic and should be repeated regularly to ensure the application continues to improve and evolve according to the needs of the users.

By continuously gathering feedback and iterating on the application, the development team can ensure that they are meeting the needs of the users and continually improving the application. This ongoing process helps maintain user satisfaction and can significantly contribute to the application's success.

6. Overall of the application

The park sharing application has been designed with the primary purpose of providing real-time availability of parking slots at various locations. The architecture of this application is composed of three fundamental pillars: the sensor parking area model, the

IP camera scenario, and the software infrastructure which consists of the backend and frontend of the application.

In the sensor parking area model, a hardware setup is utilized for real-time data collection. The principal components of this model include a TCRT5000 sensor and an ESP32 microcontroller. The TCRT5000 infrared sensor is employed to detect the presence or absence of vehicles in the parking slots. Data from the sensor is then processed and sent over a network using the ESP32 microcontroller. The integration of these components constitutes the hardware architecture of the parking sensor subsystem.

In addition to the sensor-based system, an IP camera system is also deployed. This setup uses an advanced image recognition approach that leverages AI technologies to identify parking slot occupancy. Through the implementation of techniques such as Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Region-based Convolutional Neural Networks (R-CNN, Fast R-CNN, Faster R-CNN), images are analyzed, and the presence of vehicles is identified.

The data gathered from both the sensor and IP camera subsystems is then transmitted to the backend of the application. The backend infrastructure primarily relies on Falcon API for application handling, MongoDB for data storage, and Nginx as a web server. Gunicorn, a Python WSGI HTTP server, is employed to bridge the communication between the application and the web server. This infrastructure ensures the data collected from various sources are appropriately integrated, processed, and made available for the front-end application.

The frontend of the application has been developed using React JS, a JavaScript library for building user interfaces. This section receives the processed data from the backend, translates it into a visually understandable form, and presents it to the end-users.

By using Geographic Coordinate System (GCS), the application offers real-time information about the availability of parking slots at different locations. The application's user interface is developed based on principles from Human-Computer Interaction (HCI) theories, like Cognitive Load Theory, Fitts' Law, and Distributed Cognition, to ensure an intuitive and user-friendly experience.

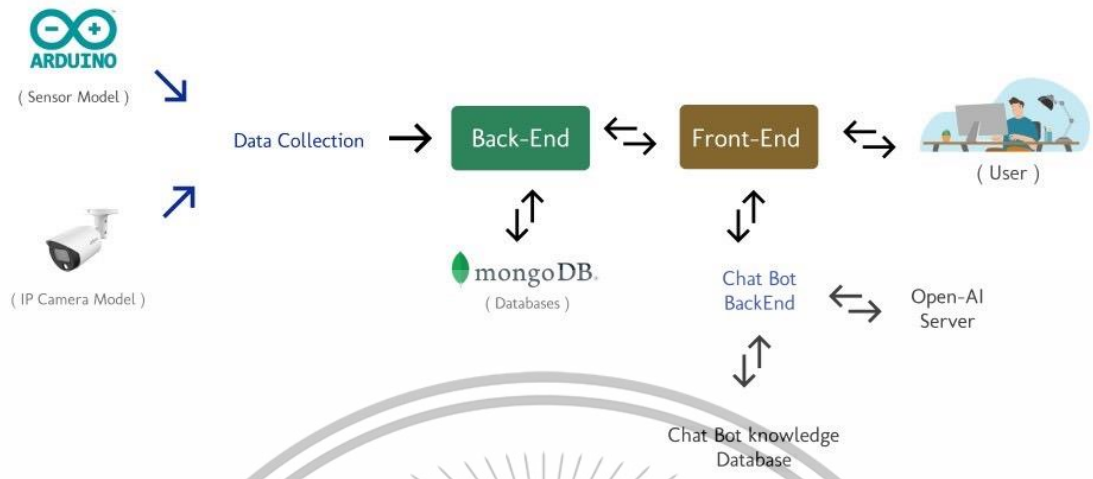


Figure 59 The overall process of the application

Lastly, the network communication between the various components of the system relies on secure and robust protocols to ensure data integrity and privacy.

In summary, the park sharing application is a comprehensive amalgamation of hardware sensors, AI-driven image detection, robust backend infrastructure, and user-centric frontend design. All these elements work cohesively to provide a real-time, user-friendly, and secure platform for sharing parking space availability.

Chapter 4 Results

1. Sensor Scenario

IR sensor model testing results by measuring the precision of the sensor, 1 means detected, and 0 means not detected by the sensor.

Table 8 Ambient light 767 lux

Ambient light (787 Lux)/ testing round	Sensor1		Sensor2		Sensor3		Sensor 4		Sensor5	
	Car in	Car out	Car in	Car out	Car in	Car out	Car in	Car out	Car in	Car out
1	1	1	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1

Precision from round Table 08:

$$Precision = \frac{TP}{TP+FP} = \frac{10+9+10}{30} = 0.967 \quad \text{--- (12)}$$

TP means the placement of the car in front of the IR sensor resulted in detection.

FP means the placement of the car in front of the IR sensor resulted in a failed detection.

Table 9 Ambient light 136 lux

Ambient light (136 Lux)/ testing round	Sensor1		Sensor2		Sensor3		Sensor 4		Sensor5	
	Car in	Car out	Car in	Car out	Car in	Car out	Car in	Car out	Car in	Car out
1	1	1	1	1	1	1	0	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1

Precision from round Table 09:

$$Precision = \frac{TP}{TP+FP} = \frac{9+10+10}{30} = 0.967 \quad \text{--- (13)}$$

TP means the placement of the car in front of the IR sensor resulted in detection.

FP means the placement of the car in front of the IR sensor resulted in a failed detection.

Table 10 Ambient light 99 lux

Ambient light (99 Lux)/ testing round	Sensor1		Sensor2		Sensor3		Sensor 4		Sensor5	
	Car in	Car out	Car in	Car out	Car in	Car out	Car in	Car out	Car in	Car out
1	0	1	1	1	1	1	0	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1

Precision from round Table 10:

$$Precision = \frac{TP}{TP+FP} = \frac{9+10+10}{30} = 0.967 \quad \text{--- (14)}$$

TP means the placement of the car in front of the IR sensor resulted in detection.

FP means the placement of the car in front of the IR sensor resulted in a failed detection.

Table 11 Ambient light 0 lux

Ambient light (0 Lux)/ testing round	Sensor1		Sensor2		Sensor3		Sensor 4		Sensor5	
	Car in	Car out	Car in	Car out	Car in	Car out	Car in	Car out	Car in	Car out
1	1	1	1	1	1	1	0	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1	1

Precision from round Table 11:

$$Precision = \frac{TP}{TP+FP} = \frac{10+10+9}{30} = 0.967 \quad \text{--- (15)}$$

TP means the placement of the car in front of the IR sensor resulted in detection.

FP means the placement of the car in front of the IR sensor resulted in a failed detection.

Overall, the

$$Precision = \left(\frac{0.967+0.967+0.967+0.967}{4}\right) \times 100 = 96.7\% \quad \text{--- (16)}$$

As observation from the mockup, the IR sensors demonstrated their functionality through consistent flashing. However, to enhance the stability and portability of the system, a decision was made to incorporate a battery as the power source instead of using the power from the computer. And continuing with the experiment of the various ambient light (lux) in the room, the fixed IR sensor detection terms of calibration achieved an impressive accuracy of 96.7%. It can be clarified that Sensor 1 is outstanding with its higher sensitivity. This differential accuracy among the sensors highlights the need for careful calibration and consideration of individual sensor characteristics. To integrate the sensor, the model ensures seamless synchronization between the physical sensors and the software components. Following the flowchart depicted in Figure 29, the sensor is continuously monitored, and the number of detected objects is determined by subtracting the count from the total number of parking spaces. This calculation provides real-time information about the availability of parking spaces. It further communicates with the server through an HTTP request, transmitting the updated parking availability information within the 1 second delay set in the program. This allows users to access accurate and real-time updates regarding parking spaces available through the server's response.

The expected outcome of the developed sensor models monitoring system encompasses accurate object detection also with ability to provide a user experience as in Figure 57. The model's successfully implemented web application offers a comprehensive and users can effortlessly access the recently updated and reliable parking availability. The sensor model has demonstrated complete and reliable monitoring capabilities, ensuring precise detection of objects in the mockup parking as shown in Figure 60, 61, and 62. By leveraging the capabilities of the IR sensors, the model effectively captures and measures the amount of reflected light, e82nabling it to detect the presence or absence of objects (demo cars).

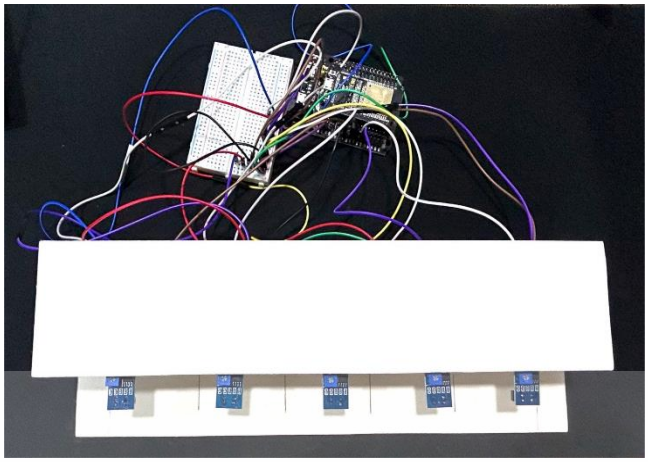


Figure 60 After assembly Sensor Model

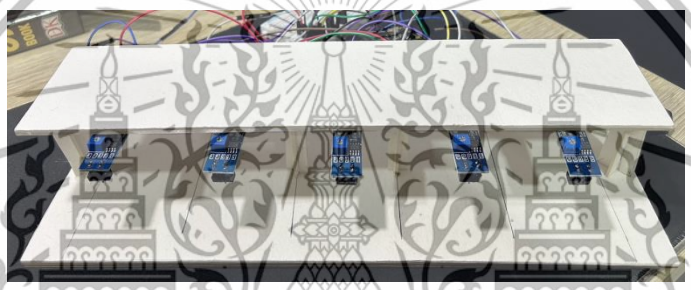


Figure 61 Sensor Model at normal state

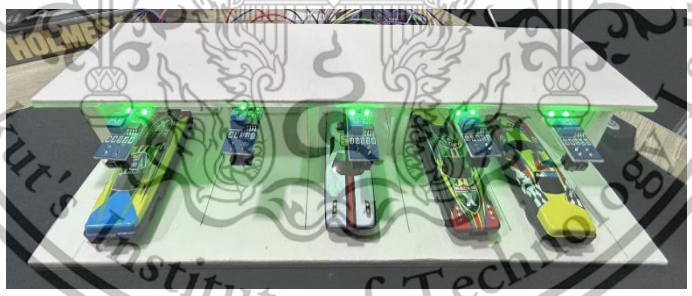


Figure 62 Sensor Model after detection

2. IP camera Scenario

After the detection, the available parking lot is calculated before passing through the website. The data will be sent to the website with this code, in figure 63.

```
availablePark = 3-NumCar
if availablePark <= 0:
    availablePark = 0
url = 'http://park-share-backend.atrobot.asia/park/update' #http://park-share.atrobot.asia/
payload = {
"id":"d59a348b-bb7f-4701-b396-9599b8c5f867",
"details":[
    {
        "name":"Front",
        "current":str(availablePark), #show available
        "total":"3"
    } ]
```

Figure 63 Send the detected data to the website

The result of the detection will be as shown in figure 64.

Figure 64 Car detection

AI Detection Observation Result

Table 12 AI Detection Accuracy Observation Result within 7 days.

Date	Period	Number of car (Actual)	Number of car (Predict)
26 April 2023	Daytime (9:00-10:00)	2	2
	Nighttime (19:00-20:00)	2	2
27 April 2023	Daytime (9:00-10:00)	2	2

	Nighttime (19:00-20:00)	1	2
28 April 2023	Daytime (9:00-10:00)	3	3
	Nighttime (19:00-20:00)	1	1
29 April 2023	Daytime (9:00-10:00)	3	3
	Nighttime (19:00-20:00)	2	2
30 April 2023	Daytime (9:00-10:00)	2	2
	Nighttime (19:00-20:00)	2	2
1 May 2023	Daytime (9:00-10:00)	1	1
	Nighttime (19:00-20:00)	2	3
2 May 2023	Daytime (9:00-10:00)	2	2
	Nighttime (19:00-20:00)	1	1

According to the data in Table 12, calculate the accuracy of the detection model by:

$$accuracy_{period\ of\ time} = \frac{actual\ and\ predict\ value\ are\ equal_{period\ of\ time}}{total\ number\ of\ cars_{period\ of\ time}} \times 100\% \quad \text{--- (17)}$$

$$accuracy_{daytime} = \frac{7}{7} \times 100 = 100\% \quad \text{--- (18)}$$

$$accuracy_{nighttime} = \frac{5}{7} \times 100 = 71\% \quad \text{--- (19)}$$

The detection accuracy of the nighttime is lower than daytime, which is around 71%. From the observation, fault detection, in both fault positive and fault negative detection, happens more often than in the daytime.

The observation data was also calculated to find an overall AI model performance in detecting the cars by using F1 score.

F1 score formula:

$$F1 = \frac{2(Precision \times Recall)}{Precision + Recall} \quad \text{--- (20)}$$

Since, True Positive (TP) of the overall is 12 times, False Positive (FP) is 2 times, and 0 in False Negative (FN).

Precision of the data:

$$Precision = \frac{TP}{TP+FP} = \frac{12}{12+2} = \frac{12}{14} = 0.86 \quad \text{--- (21)}$$

Recall of the data:

$$Recall = \frac{TP}{TP+FN} = \frac{12}{12+0} = 1.0 \quad \text{--- (22)}$$

The F1 score of the overall detection performance is:

$$F1 = \frac{2(0.86 \times 1.0)}{0.86+1.0} = 0.925 \quad \text{--- (23)}$$

F1 score can demonstrate that this AI model for car detection, accuracy is good enough for the detection because the F1 score value is near 1.

3. Application

The application developed is an innovative solution that showcases real-time parking availability per location. This chapter focuses on presenting the results achieved by the application, emphasizing the seamless integration of multiple technologies and methodologies in its design and deployment.

The application encompasses two primary scenarios: the sensor scenario and the IP camera scenario. Each one of these scenarios generates data which is meticulously analyzed and employed to reflect real-time parking availability.

The application front-end was designed using React JS, a popular and highly efficient JavaScript library. The layout and interface were crafted to maximize user experience, taking into consideration various principles of human-computer interaction such as cognitive load theory and Fitts' Law.

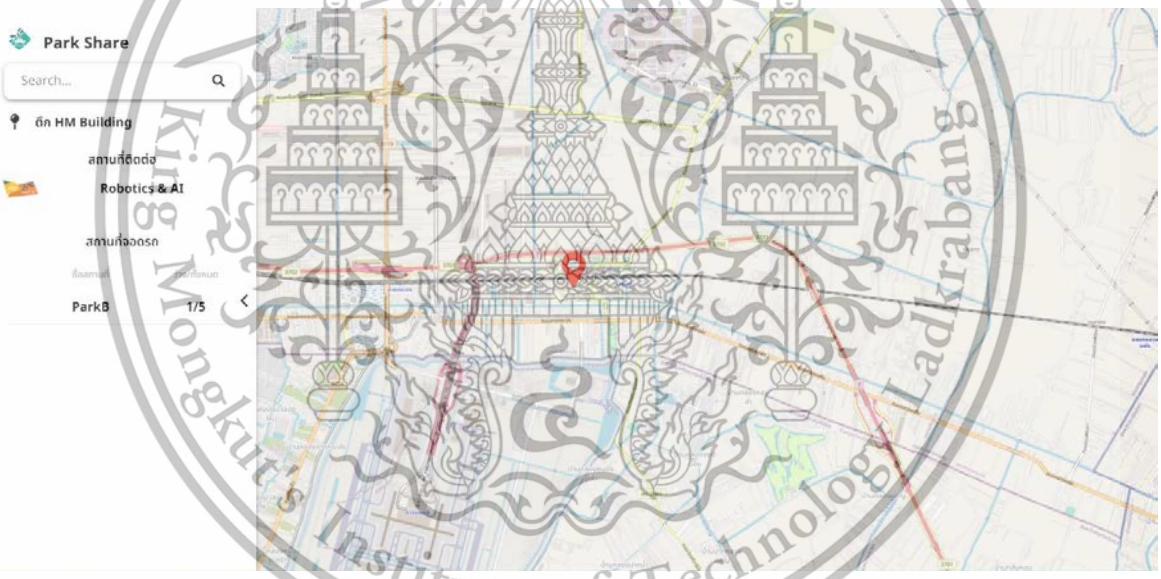


Figure 65 Website interface and integration with sensor.

It is observed that the application effectively renders the status of the parking slots in real-time, showcasing a map of the parking area with indications of filled and empty slots. The application also features a filtering option, allowing users to select specific locations of interest.

One of the most significant results is the application's performance and reliability. The back end, developed using Falcon API and MongoDB for database operations, has been robust and efficient, demonstrating excellent response times.

Moreover, the application has successfully integrated sensor data from TCRT5000 and ESP32 along with image data collected via the IP camera scenario. In the IP camera scenario, AI image detection technologies such as Convolutional Neural Networks (CNN) and Region-based Convolutional Neural Networks (RCNN) have been effectively used to detect the presence of a vehicle in a parking slot.

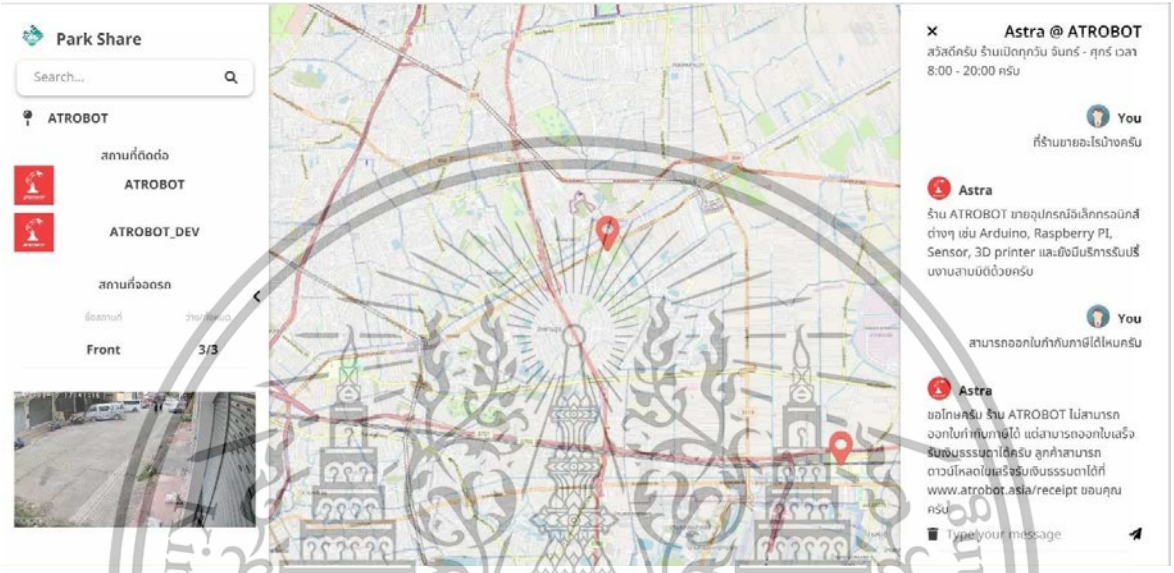


Figure 66 Website interface integrated with the sensor and IP camera.

With the use of the application, users can avoid wasting time searching for parking slots. Businesses with parking facilities can efficiently manage and monitor their parking lots, potentially increasing their overall operational efficiency.

It is important to note that the application underwent a rigorous cycle of testing and feedback collection. This iterative development approach contributed to the refinement of features and the overall improvement of the application's functionality.

In conclusion, the application serves as a comprehensive solution to the parking issue, combining sensor-based technology with AI image detection to provide real-time parking status updates. This approach not only enhances user experience but also promises a substantial improvement in the efficiency of parking management. Further studies and continued development of the application could pave the way for future innovations in smart parking solutions.

Chapter 5 Conclusion and Recommendation

The objective of this project was to develop a comprehensive parking monitoring system that includes the detection of cars by IP cameras and sensors, data transmission to a central server, and displaying the parking availability on the ParkShare web-application.

In this project, two distinct approaches have been employed to develop the application, namely the mock-up sensor parking model and the utilization of an IP camera. The sensor model utilizes infrared (IR) technology to detect the presence or availability of a car in a parking space. On the other hand, the IP camera approach incorporates image processing techniques to analyze visual data and determine the availability of a parking spot.

Through extensive experimentation under varying ambient light conditions (measured in lux), the sensor parking model has undergone rigorous calibration. This meticulous calibration process has resulted in remarkable accuracy, achieving an impressive 96.7% success rate in terms of IR sensor detection. This calibration ensures consistent and reliable performance of the sensor model, allowing it to accurately detect the availability of parking spaces even in diverse lighting environments.

In comparison, the accuracy of AI detection for outdoor parking varied depending on the period. According to Table 12: AI Detection Accuracy Observation Result within 7 Days, the system exhibited higher accuracy during the daytime compared to the nighttime. It is important to note that the accuracy of AI detection can be influenced by various factors, and consistent monitoring and evaluation are necessary to ensure reliable performance.

Overall, the sensor-based indoor parking detection system proved to be more accurate and reliable for the purpose of parking detection, contributing to the satisfaction of users utilizing the ParkShare. The accuracy of AI detection during the daytime period is around 100%, but for the nighttime period it is only around 71%.

Further development in AI detection can be developed by receiving the global time and determining the period during the day. Compare 2 times, which are global time and determined time, if the detection happens during the daytime, the confidence threshold can be set at 0.5, but if the detection happens during the nighttime, the confidence threshold can be set in higher value of the confidence threshold such as set at 0.6 or 0.7.

APPENDIX

APPENDIX A

PROGRAMMING IN SENSOR MODEL

Using Arduino to get the https post to connect with the web application that we created.

1.Include the relevant libraries in your code WiFi, HTTPClient, and JSON libraries:

```
#include <Arduino_JSON.h>
#include <WiFi.h>
#include <HTTPClient.h>
```

2.Declares an integer variable and number of values. This variable will be used for the digital port and represent the pin number connected to the detector. Including storing the sensor value readings from the corresponding pins and calibration value for the sensors.

```
int detectorPin1 = 19;
int detectorPin2 = 18;
int detectorPin3 = 5;
int detectorPin4 = 4;
int detectorPin5 = 17;  // obstacle avoidance sensor interface
int val1, val2, val3, val4, val5;
int cal;
int total = 20;         // variable to store result
```

3.Set identifier name and pin of the Wi-Fi network.

```
const char* ssid = "tkk";
const char* password = "12345678";
```

4.Make the HTTP POST to the server. Used to keep track of the time between request initially to set to be zero and hold the time delay to 1000 milliseconds (or 1 second)

```
// Domain Name with full URL Path for HTTP POST Request
const char* serverName = "http://park-share-
backend.atrobot.asia/park/update";

// THE DEFAULT TIMER IS SET TO 10 SECONDS FOR TESTING PURPOSES
// For a final application, check the API call limits per hour/minute
to avoid getting blocked/banned
unsigned long lastTime = 0;
// Set timer to 10 minutes (600000)
// Timer set to 10 seconds (10000)
unsigned long timerDelay = 1000;
```

1. Setup all variables for initializing the serial communication with a baud rate of 115200.

```
void setup() {
  Serial.begin(115200);

  pinMode(detectorPin1, INPUT);  // Define obstacle avoidance sensor
as input interface
  pinMode(detectorPin2, INPUT);
  pinMode(detectorPin3, INPUT);
  pinMode(detectorPin4, INPUT);
  pinMode(detectorPin5, INPUT);
```

```
  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());
  Serial.println("Timer set to 10 seconds (timerDelay variable),
it will take 10 seconds before publishing the first reading.");
```

2. Loop function to read and calculate the values from the five obstacle sensors.

```
void loop() {
  //read value from sensor
  val1 = digitalRead(detectorPin1); // Read value from sensor
  val2 = digitalRead(detectorPin2);
  val3 = digitalRead(detectorPin3);
  val4 = digitalRead(detectorPin4);
  val5 = digitalRead(detectorPin5);
  Serial.println(val1);
  Serial.println(val2);
  Serial.println(val3);
  Serial.println(val4);
  Serial.println(val5);
  cal = val1 + val2 + val3 + val4 + val5;
  Serial.println("here");
  Serial.println(cal);
  delay (200);
```

3. Connection to a Wi-Fi network uses the ESP32 Wi-Fi module and then periodically sends an HTTP POST request to a server specified by the serverName variable. And checking elapsed time since the last request specified delay.

```
//Send an HTTP POST request every 10 seconds
  if ((millis() - lastTime) > timerDelay) {
    //Check WiFi connection status
    if(WiFi.status()== WL_CONNECTED){
      WiFiClient client;
      HTTPClient http;
http.begin(client, serverName);
```

4. Add the header to the HTTP request. Especially the format of the data being sent in the request body, which is JSON.

```
  http.addHeader("Content-Type", "application/json");
      // JSON data to send with HTTP POST
      //no api key
```

5. Creating a JSON formatted string that will be sent in the body of the HTTP POST request.

```
   String httpRequestData = "{\"id\":\"a8933358-f3dc-4699-9793-
a085763a7d2e\",\"details\":[{\"name\":\"ParkB\",\"current\":\"" +
String(cal) + "\",""\"total\":\"5\"}]}";
```

6. Send HTTP POST request.

```
int httpResponseCode = http.PUT(httpRequestData);

      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);

      String payload = http.getString();
      Serial.println(payload);
      // Free resources
      http.end();
    }
    else {
      Serial.println("WiFi Disconnected");
    }
    lastTime = millis();
  }
}
```

APPENDIX B

PROGRAMMING IN AI OBJECT DETECTION

Separate into 2 files: for detecting and for displaying.

A. For detecting:

```
import tensorflow as tf
import cv2

modelPath = r'D:\kmitl\capstone\TFODCourse\Tensorflow\workspace\pre-
trained-models\faster_rcnn_resnet50_v1_640x640_coco17_tpu-8\saved_model'

#---Load the pre-trained model---
model = tf.saved_model.load(modelPath)

#---detection region---
start_x = 300
start_y = 300
end_x = 1700
end_y = 300

#---Convert the image to a tensor---
def detect_car(frame,cvtframe):
    input_tensor = tf.convert_to_tensor(cvtframe)
    input_tensor = input_tensor[tf.newaxis, ...]

    #---Run the inference on GPU---
    with tf.device('/job:localhost/replica:0/task:0/device:GPU:0'):
        detections = model(input_tensor)
        #print(detections['detection_classes'][0])

    #---Draw the bounding boxes---
    count = 0 #count detected cars
    frame = cv2.line(frame, (start_x, start_y), (end_x,end_y), (0,0,255),
5)
    for i in range(len(detections['detection_scores'][0])):
        score = detections['detection_scores'][0][i]
        if score > 0.5 and detections['detection_classes'][0][i] == 3: #3
is car/ night score>0.18/day score>0.5
            score = "{:.2f}".format(score)
            displayText = '{}:{}'.format('car',score)
            bbox = detections['detection_boxes'][0][i]
            ymin, xmin, ymax, xmax = bbox
            h, w, _ = frame.shape
            left, right, top, bottom = int(xmin * w), int(xmax * w),
int(ymin * h), int(ymax * h)
            area = abs(left-right)*abs(top-bottom)

            if area>50000 and area<1000000:
            # print(bottom, start_y)
```

```
                if bottom > start_y:
                    count += 1

                    cv2.rectangle(frame, (left, top), (right, bottom), (0,
255, 0), 2)
                    cv2.putText(frame, displayText, (left, top-10),
cv2.FONT_HERSHEY_COMPLEX, 1, (255,0,0), 1)

    print(frame.shape)
    #print('top:',top)
    #print('bottom:', bottom)
    #print('left:', left)
    #print('right:', right)
    print('Number of parked car: ',count)
    return count
```

B. For displaying:

```
import cv2
import time
from threading import Thread # library for implementing multi-threaded
processing
from detection import detect_car
import requests
import os

# defining a helper class for implementing multi-threaded processing
class VDOStream :
    #stream_id="rtsp://atrobot_cam:atrobot_cam@atrobot.thddns.net:8180/stre
am1"
    def __init__(self, stream_id):
        self.stream_id = stream_id

        # opening video capture stream
        self.vcap = cv2.VideoCapture(self.stream_id)
        if self.vcap.isOpened() is False :
            print("[Exiting]: Error accessing webcam stream.")
            exit(0)
        fps_input_stream = int(self.vcap.get(5))
        #print("FPS of webcam hardware/input stream:
{}".format(fps_input_stream))

        # reading a single frame from vcap stream for initializing
        self.grabbed , self.frame = self.vcap.read()
        if self.grabbed is False :
            print('[Exiting] No more frames to read')
            exit(0)

        # self.stopped is set to False when frames are being read from
self.vcap stream
        self.stopped = True

        # reference to the thread for reading next available frame from
input stream
        self.t = Thread(target=self.update, args=())
        self.t.daemon = True # daemon threads keep running in the
background while the program is executing

    # method for starting the thread for grabbing next available frame in
input stream
    def start(self):
        self.stopped = False
        self.t.start()

    # method for reading next frame
```

```
    def update(self):
        while True :
            if self.stopped is True :
                break
            self.grabbed , self.frame = self.vcap.read()
            if self.grabbed is False :
                print('[Exiting] No more frames to read')
                self.stopped = True
                break
        self.vcap.release()

    # method for returning latest read frame
    def read(self):
        return self.frame

    # method called to stop reading frames
    def stop(self):
        self.stopped = True


# initializing and starting multi-threaded webcam capture input stream
stream_id="rtsp://atrobot_cam:atrobot_cam@atrobot.thddns.net:8180/stream1"
webcam_stream = VDOStream(stream_id) # stream_id = 0 is for primary camera
webcam_stream.start()

# processing frames in input stream
num_frames_processed = 0
start = time.time()

while True :
    if webcam_stream.stopped is True :
        break
    else :
        frame = webcam_stream.read()


    h,w,_=frame.shape
    vdo_h = int(h/2)
    vdo_w =int(w/2)

    #grayscale
    frame_gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    frame_gray = cv2.cvtColor(frame_gray,cv2.COLOR_GRAY2BGR)

    # adding a delay for simulating time taken for processing a frame
    delay = 0.03 # delay value in seconds. so, delay=1 is equivalent to 1
second
    time.sleep(delay)
    num_frames_processed += 1
```

```
    NumCar = detect_car(frame,frame_gray)
    resizeframe = cv2.resize(frame, (vdo_w,vdo_h))
    #resizeframe = cv2.line(resizeframe, (250, 180), (700,210), (0,0,255),
2)
    #cv2.imwrite('D:\kmitl\capstone\TFtry1\img'+'\\'+str(num_frames_process
ed)+'.jpg',frame)
    cv2.imshow('frame' , resizeframe)

    if cv2.waitKey(1) == ord('q'):
        break

    availablePark = 3-NumCar
    if availablePark <= 0:
        availablePark = 0
    url = 'http://park-share-backend.atrobot.asia/park/update'
#http://park-share.atrobot.asia/
    payload = {
    "id":"d59a348b-bb7f-4701-b396-9599b8c5f867",
    "details":[
        {
            "name":"Front",
            "current":str(availablePark), #show available
            "total":"3"
        } ]
}
    response = requests.put(url, json=payload)
end = time.time()
webcam_stream.stop() # stop the webcam stream

# closing all windows
cv2.destroyAllWindows()
```

APPENDIX C

PROGRAMMING IN BACKEND

A. Setup Server Hardware

Since the website runs in Ubuntu OS which is in a virtual machine. These following are the from scratch setup.

1.)Prepare Empty Server

Any computer which has enough capability to be the host or follow the host specification in 3.4.1.2.1.

2.)Install XCP-NG OS

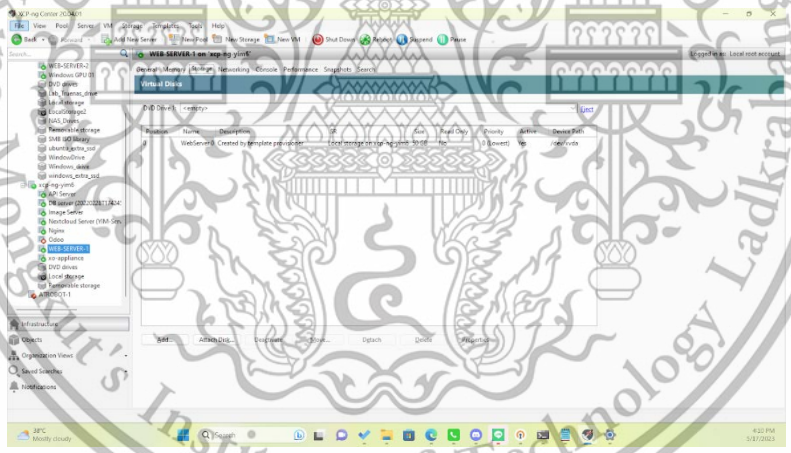
Download the ISO from the official website [https://xcp-ng.org/]. And install it using USB using RUFUS.

3.)Setup Static IP

This setup is used to prevent IP changing.

4.)Create Virtual Machine and Install Ubuntu Server

Use XCP-ng Center to access the server. The XCP-ng Center provides a user interface. This software is used to create virtual machines. Once the virtual machine has been created. It will automatically boot up the guest OS. Follow the instructions to install the OS as normal.



B. Setup Server VM Software

Once the virtual machine is up and running. Since the ubuntu server doesn't have the interface. The way to access the machine is to use SSH.

1.) Access via SSH

Find and get the guest OS IP address and use SSH to login to the server.

2.) Install NGINX

Using this command to install nginx on ubuntu server.

‘’’

sudo apt install nginx

‘’’

Then start and automatic start using this command.

‘’’

sudo systemctl start nginx.

sudo systemctl enable nginx.

‘’’

C. Setup Server Network

1.) Register Network To DDNS

Since The network provider doesn’t have private IP. DDNS is necessary to handle the change of the IP. The AIS network provider is a partner with THDDNS.

Register the THDDNS via the link below and it will automatically link with the home network.

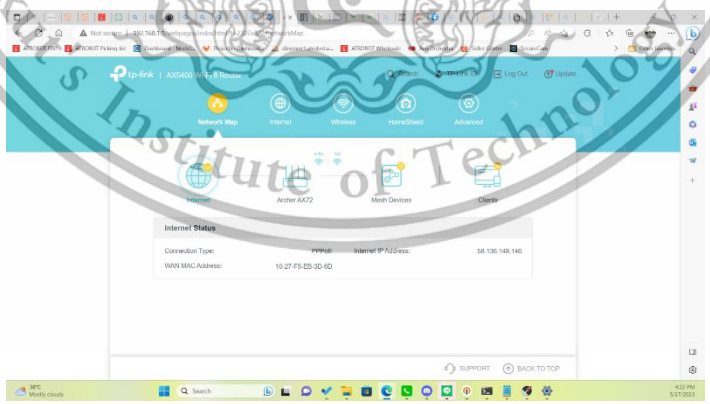
https://www.thddns.net/login

2.)Forward the router port to be accessible via internet.

The internet and the server are not linked by default. To link the internet with the server, a forward port is needed.

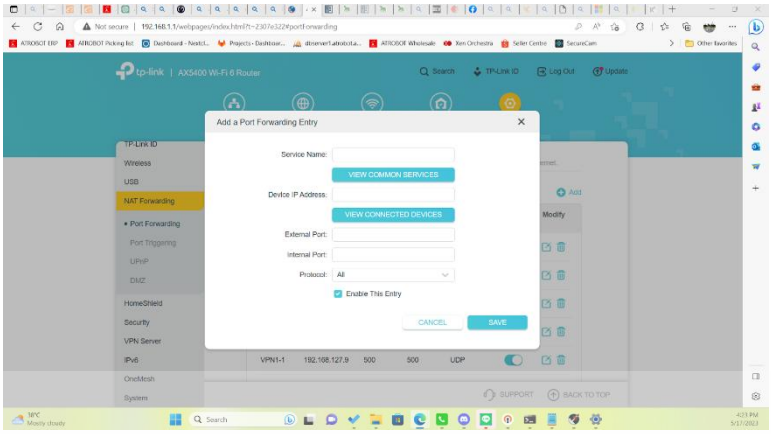
2.1.) Login to the router

The router user interface is usually located at IP 192.168.1.1 Go to the IP and login.



2.2.) Forward port

Locate the forward port function.



The interface will be varying depend on the router model but usually will have the following input information which are:

Service Name: Name of the service. It can be anything.

Device IP Address: The IP address of the computer that hosts the website or backend.

External Port: The port that the internet user will use. It can be any port but not the same as the defined port. It can be the same with an internal port.

Internal Port: The port that the server uses in the internal network. For example, http would use 80.

D. Create the REST API

Software Structure

1.) Components folder

The components folder will store the features of each function which are park and parks.

1.1) components/park folder

This folder will have the python files which are park.py and utils.py.

a. components/park/park.py

The logic for each endpoint for individual parks.

Import Statements

```
import json
import falcon
import components.park.utils as utils
import standard.http as http_standard
import standard.park as park_standard
from cerberus import Validator
```

These few lines of code are Import Statements for different modules that are used in the code.

Schema Definitions

```
park_update_schema = {
    park_standard.ID: {'type': 'string', 'required': True},
    park_standard.DETAILS: {'type': 'list', 'required': True,
'schema': {
        'type': 'dict', 'schema': {
            park_standard.NAME: {'type': 'string', 'required': True},
            park_standard.CURRENT: {'required': True},
            park_standard.TOTAL: {'required': True}
        }}
    }
}
```

The next part defines validation schemas for park updates and park creation. These schemas are used to validate the data received in the HTTP request. They're using the Cerberus library for validation.

Validator Initialization

```
validator = Validator()
```

The next line initializes a Validator instance from the Cerberus library which is used later for validating data.

ParkUpdate Class

```
class ParkUpdate:
    def on_put(self, req, resp):
        raw_data = req.bounded_stream.read()
        data = json.loads(raw_data.decode('utf-8'))
        if not validator.validate(data, park_update_schema):
            resp.status = falcon.HTTP_400
            resp.body = json.dumps({
                http_standard.STATUS: http_standard.STATUS_FAIL,
                http_standard.MESSAGE: validator.errors
            })
            return
        result = utils.update_park(data['id'], data['details'])
        if result[http_standard.STATUS] == http_standard.STATUS_FAIL:
            resp.status = falcon.HTTP_400
            resp.body = json.dumps({
                http_standard.STATUS: http_standard.STATUS_FAIL,
                http_standard.MESSAGE: result[http_standard.MESSAGE]
            })
            return

        resp.status = falcon.HTTP_200
        resp.body = json.dumps(
            {http_standard.MESSAGE: 'Park updated successfully'})

park_create_schema = {
```

```
    park_standard.NAME: {'type': 'string', 'required': True},
    park_standard.LOCATION: {'type': 'dict', 'required': True,
'schema': {
        park_standard.LAT: {'required': True},
        park_standard.LNG: {'required': True}
    }},
}
```

This class has a method on_put which is called when a PUT HTTP request is made. This method reads data from the request, validates it using the park_update_schema, and calls a method from utils to update the park. If any error occurs during the process, the method responds with an HTTP 400 status and an error message. If the update is successful, it responds with an HTTP 200 status and a success message.

ParkCreate Class

```
class ParkCreate:
    def on_post(self, req, resp):
        raw_data = req.bounded_stream.read()
        data = json.loads(raw_data.decode('utf-8'))
        if not validator.validate(data, park_create_schema):
            resp.status = falcon.HTTP_400
            resp.body = json.dumps({
                http_standard.STATUS: http_standard.STATUS_FAIL,
                http_standard.MESSAGE: validator.errors
            })
            return
        result = utils.create_park(data['name'], data['location'])
        if result[http_standard.STATUS] == http_standard.STATUS_FAIL:
            resp.status = falcon.HTTP_400
            resp.body = json.dumps({
                http_standard.STATUS: http_standard.STATUS_FAIL,
                http_standard.MESSAGE: result[http_standard.MESSAGE]
            })
            return

        resp.status = falcon.HTTP_200
        resp.body = json.dumps(
            {
            http_standard.STATUS: http_standard.STATUS_SUCCESS,
            http_standard.MESSAGE: 'Park created successfully',
            park_standard.ID: result[park_standard.ID]
            })
```

This class has a method on_post which is called when a POST HTTP request is made. This method reads data from the request, validates it using the park_create_schema, and calls a method from utils to create a park. If any error occurs during the process, the method responds with an HTTP 400 status and an error message. If the park creation is successful, it responds with an HTTP 200 status, a success message, and the ID of the newly created park.

ParkGet Class

```
class ParkGet:
    def on_get(self, req, resp):
        id = req.get_param('id')
        if id == None:
            resp.status = falcon.HTTP_400
            resp.body = json.dumps({
                http_standard.STATUS: http_standard.STATUS_FAIL,
                http_standard.MESSAGE: 'Bad Request. Please provide
id'
            })
            return
        park = utils.get_park(id)
        if park[http_standard.STATUS] == http_standard.STATUS_FAIL:
            resp.status = falcon.HTTP_400
            resp.body = json.dumps({
                http_standard.STATUS: http_standard.STATUS_FAIL,
                http_standard.MESSAGE: park[http_standard.MESSAGE]
            })
            return
        resp.status = falcon.HTTP_200
        resp.body = json.dumps(park)
```

This class has a method on_get which is called when a GET HTTP request is made. This method reads an 'id' parameter from the request, checks if it's provided, and then calls a method from utils to get the park by ID. If any error occurs during the process, the method responds with an HTTP 400 status and an error message. If the park retrieval is successful, it responds with an HTTP 200 status and the retrieved park's data.

b. components/park/utils.py

This file will contain a helper function to help with park.py for better readability.

Import Statements

```
import utils.database as db
import standard.http as http_standard
import standard.park as park_standard
import uuid
```

The code begins with import statements for different modules that are used in the code. It imports utils.database as db, standard.http as http_standard, standard.park as park_standard, and uuid.

Park Collection

```
park_collection = db.park_collection
```

The park_collection is a collection in a MongoDB database, as can be inferred from the find_one, update_one, and insert_one method used later in the code. This collection presumably contains data about parks.

Function update_park

```
def update_park(id: str, details: list) -> dict:
    # Check if park exist
    target_park = park_collection.find_one({park_standard.ID:id})
    if target_park == None:
        return {
            http_standard.STATUS:http_standard.STATUS_FAIL,
            http_standard.MESSAGE:'Bad Request. Park not found.
Please check your id'
            }
    # Update park details
    park_collection.update_one({'id':id},{'$set':{park_standard.DETAI
LS:details}})
    return {
        http_standard.STATUS:http_standard.STATUS_SUCCESS,
        http_standard.MESSAGE:'Park updated successfully'
        }
```

This function takes an id and details as arguments and attempts to update a park in the park_collection. It first checks if the park with the given id exists. If the park doesn't exist, it returns a dictionary with status as 'fail' and an error message. If the park does exist, it updates the park details and returns a dictionary with status as 'success' and a success message.

Function create_park

```
def create_park(name: str, location: dict) -> dict:
    # Check if park exist
    target_park = park_collection.find_one({park_standard.NAME:name})
    if target_park != None:
        return {
            http_standard.STATUS:http_standard.STATUS_FAIL,
            http_standard.MESSAGE:'Bad Request. Park already exist.
Please check your name'
            }
    # Create new park
    new_id = str(uuid.uuid4())
    new_park = {
        park_standard.ID: new_id,
        park_standard.NAME: name,
        park_standard.LOCATION: location,
        park_standard.DETAILS: []
    }
    park_collection.insert_one(new_park)
    return {
        http_standard.STATUS:http_standard.STATUS_SUCCESS,
        http_standard.MESSAGE:'Park created successfully',
        park_standard.ID: new_id
```

```
        }
```

This function takes a name and location as arguments and attempts to create a new park in the park_collection. It first checks if a park with the given name already exists. If the park exists, it returns a dictionary with status as 'fail' and an error message. If the park doesn't exist, it creates a new park with a unique id, the given name and location, and an empty details list. Then, it inserts the new park into the park_collection and returns a dictionary with status as 'success', a success message, and the new park's id.

Function get_park

```
def get_park(id:str) -> dict:
    result = park_collection.find_one({park_standard.ID:id})
    if result == None:
        return {
            http_standard.STATUS:http_standard.STATUS_FAIL,
            http_standard.MESSAGE:'Bad Request. Park not found.
Please check your id'
            }
    # Remove _id field
    result.pop('_id')
    return {
        http_standard.STATUS:http_standard.STATUS_SUCCESS,
        http_standard.MESSAGE:'Park retrieved successfully',
        http_standard.DATA: result
        }
```

This function takes an id as argument and attempts to retrieve a park from the park_collection. It first checks if the park with the given id exists. If the park doesn't exist, it returns a dictionary with status as 'fail' and an error message. If the park does exist, it removes the '_id' field from the park data (a default field added by MongoDB) and returns a dictionary with status as 'success', a success message, and the park data.

1.2) components/parks folder

This folder will have the python files which are parks.py and utils.py.

a. components/parks/parks.py

Import Statements

```
import json
import falcon
import components.parks.utils as utils
import standard.http as http_standard
import standard.park as park_standard
from cerberus import Validator
```

The first few lines are importing necessary modules for the code. json is used for handling JSON data, falcon is a minimalist WSGI library for building web APIs, components.parks.utils is likely a module containing utility functions related to parks (in this case, get_parks()), standard.http and standard.park are modules that contain standard constant definitions for the project, and cerberus is a validation library used for data validation.

ParksGet Class

This class is a resource handler for a specific route in a Falcon web application.

```
class ParksGet:
    def on_get(self, req, resp):
        parks = utils.get_parks()
        resp.status = falcon.HTTP_200
        resp.body = json.dumps(parks)
```

on_get(self, req, resp): This method handles HTTP GET requests. When a GET request is made to the route associated with the ParksGet resource, this method is called. Here's what it does:

parks = utils.get_parks(): This line calls the get_parks() function from the utils module to retrieve all parks data. It's assumed that this function interacts with a database or some data source to fetch the parks data and returns it.

resp.status = falcon.HTTP_200: This line sets the HTTP status of the response to 200, which means "OK" or successful.

resp.body = json.dumps(parks): This line sets the body of the HTTP response to a JSON string representation of the parks data. json.dumps() is used to convert the Python object into a JSON string.

b. components/parks/utils.py

This file contains the helper function that will be used in parks.py.

Import Statements

```
import utils.database as db
import standard.http as http_standard
import standard.park as park_standard
import uuid
```

The first few lines are import statements for different modules that are used in the code. It imports utils.database as db, standard.http as http_standard, standard.park as park_standard, and uuid.

Park Collection

```
park_collection = db.park_collection
```

The park_collection is a collection in a MongoDB database, as can be inferred from the find method used later in the code. This collection presumably contains data about parks.

Function get_parks

```
def get_parks() -> dict:
    # Check if park exist
    all_parks = park_collection.find()
    parks = []
    for park in all_parks:
        parks.append({
            park_standard.ID: park[park_standard.ID],
            park_standard.NAME: park[park_standard.NAME],
            park_standard.LOCATION: park[park_standard.LOCATION]
        })
    return {
        http_standard.STATUS:http_standard.STATUS_SUCCESS,
        http_standard.MESSAGE:'Parks retrieved successfully',
        park_standard.PARKS: parks
        }
```

This function retrieves all parks from the park_collection. Here's what it does in detail:

all_parks = park_collection.find(): This line retrieves all documents (parks) in the park_collection. The find method without any parameters returns all documents in the collection.

parks = []: This line initializes an empty list to store the retrieved parks.

The for loop iterates over all retrieved park documents. For each park, it creates a dictionary with the park's id, name, and location, and appends this dictionary to the parks list.

Finally, the function returns a dictionary with status as 'success', a success message, and the parks list. The keys used in the dictionary are presumably defined as constants in the http_standard and park_standard modules.

2.) standard folder

a. standard/http.py

```
STATUS = 'status'
STATUS_FAIL = 'fail'
STATUS_SUCCESS = 'success'
STATUS_ERROR = 'error'
MESSAGE = 'message'
DATA = 'data'
```

This piece of Python code defines a series of string constants. These constants are often used in a program to avoid hardcoded strings and to make the code more maintainable. Each constant represents a specific string that might be used frequently throughout the program.

b. standard/park.py

```
# Park collection structure:
ID = 'id'
DETAILS = 'details'
NAME = 'name'
AVAILABLE = 'available'
TOTAL = 'total'
LOCATION = 'location'
LAT = 'lat'
LNG = 'lng'
PARKS = 'parks'
CURRENT = 'current'

# Example
structure = {
    ID: 'id',
    DETAILS: [
        {
            NAME: 'ParkA',
            AVAILABLE: 0,
            TOTAL: 0
        }
    ],
    NAME: 'KMITL',
    LOCATION: {
        LAT: 13.724,
        LNG: 100.493
    },

}
```

Constant Definitions: The code begins by defining a series of string constants, each representing a key in the park collection structure.

Example Structure: The structure dictionary is an example of how a park record should be structured. Here's what each key-value pair represents:

ID: A string that represents the unique identifier for the park.

DETAILS: A list of dictionaries where each dictionary contains information about a specific park.

NAME: A string that represents the name of the park.

AVAILABLE: A number representing the available spaces or facilities at the park.

TOTAL: A number representing the total spaces or facilities in the park.

NAME: A string that represents the name of the park collection or area (in this example, 'KMITL').

LOCATION: A dictionary that represents the geographical location of the park.

LAT: A number representing the latitude of the park's location.

LNG: A number representing the longitude of the park's location.

3.) Utils folder

The folder contains an extra utils file.

a. **utils/database.py**

This Python code is responsible for setting up a connection to a MongoDB database and defining a specific collection within that database. Here is a breakdown of the different sections:

Import Statements

```
from pymongo import MongoClient
import config
```

The first two lines are import statements for the necessary Python libraries. pymongo is a Python driver for MongoDB and config is likely a Python module in the same project that contains configuration details.

Creating a MongoDB Client

```
client = MongoClient(config.DB_ADDRESS, 27017)
```

The client variable is an instance of MongoClient, which is a client-side representation of a MongoDB cluster. MongoClient takes two parameters: the first one is the address of the MongoDB server (retrieved from config.DB_ADDRESS), and the second one is the port number (27017 is the default MongoDB port).

Accessing a MongoDB Database

```
db = client[config.DATABASE_NAME]
```

db is a database instance within the MongoDB client. client[config.DATABASE_NAME] accesses the database with the name specified in config.DATABASE_NAME.

Accessing a MongoDB Collection

```
park_collection = db['park']
```

park_collection is a collection within the db database. Collections in MongoDB are analogous to tables in relational databases. They contain documents (analogous to rows in a table), which are the basic units of data in MongoDB. db['park'] accesses the collection named 'park' within the db database.

4.) app.py

The main code of the app.

Import Statements

```
import falcon
import components.park.park as park
import components.parks.parks as parks
```

The first few lines are import statements for the necessary Python libraries and modules. falcon is a Python library for building web APIs. components.park.park and components.parks.parks are likely modules in the same project that contain classes for handling different API endpoints.

Creating a Falcon App

```
app = falcon.App(cors_enable=True)
```

The app variable is an instance of a Falcon web app. falcon.App(cors_enable=True) creates a new Falcon web app with CORS (Cross-Origin Resource Sharing) enabled. CORS is a mechanism that allows many resources (e.g., fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain from which the resource originated.

Adding Routes to the App

```
app.add_route('/park/update', park.ParkUpdate())
app.add_route('/park/create', park.ParkCreate())
app.add_route('/park/get', park.ParkGet())
app.add_route('/parks/get', parks.ParksGet())
```

The app.add_route() method is used to define routes for the web app. A route is a URL pattern that is used to find the appropriate resource for a given HTTP request. Each route is associated with a class that handles requests to that route. For example, app.add_route('/park/update', park.ParkUpdate()) defines a route for the URL /park/update that is handled by the ParkUpdate class in the park module.

Running the App

```
if __name__ == '__main__':
    from wsgiref import simple_server
    httpd = simple_server.make_server('0.0.0.0',8000, app)
    print('Serving on port 8000...')
    httpd.serve_forever()
```

The code inside the if __name__ == '__main__': block is executed when the script is run directly (not imported as a module). This code creates a WSGI server that serves the Falcon app. wsgiref.simple_server is a simple WSGI (Web Server Gateway Interface) server implementation that comes with Python. simple_server.make_server('0.0.0.0',8000, app) creates a new server that listens on all available network interfaces ('0.0.0.0') and port 8000. The server is set to serve the Falcon app. httpd.serve_forever() starts the server and makes it begin listening for incoming HTTP requests.

5.) config.py

```
DB_ADDRESS = '192.168.1.199'
DATABASE_NAME = 'ParkShare'
```

DB_ADDRESS = '192.168.1.199': This line defines a constant DB_ADDRESS which represents the address of the database server. In this case, the server is located at the IP address 192.168.1.199.

DATABASE_NAME = 'ParkShare': This line defines a constant DATABASE_NAME which represents the name of the database. In this case, the name of the database is ParkShare.

6.) Run the Rest API Code on the target server.

a. Login to the target server via SSH

b. Put the source code in the home directory.

c. Create the python environment.

‘’’

python3 -m venv env

‘’’

Install the package (if necessary)

‘’’

sudo apt install python3-venv

‘’’

d. Source the virtual environment and install python library.

‘’’

source env\Script\activate

pip install -r requirement.txt

‘’’

e. Create a service to automatic run the code.

[Unit]

Description=gunicorn daemon

After=network.target

[Service]

User=atrobot

Group=atrobot

WorkingDirectory=/home/atrobot/park-share/backend

ExecStart=/home/atrobot/park-share/env/bin/gunicorn app:app -b 0.0.0.0:8003

Restart=always

[Install]

WantedBy=multi-user.target

f. Save the service as park-share.service

g. Move the park-share.service to /etc/systemd/system/ using the follow command

‘’’

sudo mv park-share.service /etc/systemd/system/park-share.service

‘’’

h. Start the application and enable the auto start.

‘’’

sudo systemctl start park-share.service

sudo systemctl enable park-share.service

‘’’

APPENDIX D

PROGRAMMING IN FRONT END

A. Install Node JS

Download the installer from https://nodejs.org/en/download and install to the system

1.Create the React App

After install create, Launch the command prompt and create react app using this command.

‘’’

npx create-react-app my-app

‘’’

2.Run the application.

‘’’

npm start

‘’’

B. Develop the front end.

Frontend Project structure

The structure of the react app has 3 main folders which are API, components, and page. The API folder will contain all the files related to the API. Such as endpoint URL method The components folder contains a piece of code that define what component on the screen should look like

The page folder will define what components should be on the page.

C. Example of the file in API folder

Imports

```
import axios from 'axios';
import * as config from '../config';
```

These two lines import two separate modules.

AXIOS is a popular, promise-based HTTP client that works both in the browser and in a Node.js environment. It provides a single API for dealing with XMLHttpRequests and node's HTTP interface.

config is a custom module, presumably located one directory up from the current file. The import * syntax means it imports all exported members from the 'config' module. This is likely where the backend URL and other configuration data is stored.

getPark function

```
export const getPark = async (id) => {
    const params = {
        id: id
    };
    const response = await
axios.get(`${config.BACKEND_URL}/park/get`, {params});
    return response.data;
};
```

This is an asynchronous function named getPark that's being exported from this module. It takes one argument, id, which presumably is the identifier for a specific park.

Inside the function, an object named params is being created. This object has a property id, which is given the value of the function's id parameter.

Then, an HTTP GET request is made using axios. The URL for this request is constructed using a template string to concatenate the BACKEND_URL value from the config module with the path /park/get. The params object is included in the options object as the params property, which means it will be sent as query parameters with the GET request.

Since this is an asynchronous function, the await keyword is used to pause execution of the function until the axios.get() promise resolves. The resolved value of the promise (the response from the server) is then assigned to the response variable.

The function then returns response.data, which is the body of the HTTP response. Depending on how the server is set up, this could be any type of data, but it's often an object or an array of objects in JSON format.

D. Example of the file in components folder

```
import React, { useEffect, useState } from 'react'
import './LeftPanel.scss'
import '../../scss/font.scss'
import * as responsive from '../../jsx/responsive'
import * as park_api from '../../api/park'
import Parkinglist from './sub-components/ParkingList/ParkingList'
import ContactList from './sub-components/ContactList/ContactList'
import ParkName from './sub-components/ParkName/ParkName'
import CameraFeed from './sub-components/CamFeed/CamFeed'
import { useRef } from 'react'

export default function LeftPanel({ setTargetResult, targetResult,
targetResultRef, show, setCurrent_bot_agent }) {
    const [parkData, setParkData] = useState(null);
    const [camDetail, setCamDetail] = useState({ mac:
'9C:53:22:0B:46:1D', name: 'Front' });
    useEffect(() => {
        loadData();
    }, [targetResult]);

    // Load data every 1 seconds
    useEffect(() => {
        const interval = setInterval(() => {
            loadData();
        }, 1000);
        return () => clearInterval(interval);
    }, []);

    const loadData = () => {
        const id = targetResultRef.current?.id;
        if (id) {
            park_api.getPark(id).then((response) => {
                setParkData(response['data']);
            });
        }
    }

    const [width, setWidth] = useState(window.innerWidth);

    useEffect(() => {
        const handleResize = () => setWidth(window.innerWidth);
        window.addEventListener('resize', handleResize);
        return () => {
            window.removeEventListener('resize', handleResize);
        };
    }, []);

    return (
        <div className='left-panel-component'
            style={{
                left: ((width > responsive.WIDTH_MOBILE) ? (show ?
'0' : '-100%') : '0'),
                top: ((width <= responsive.WIDTH_MOBILE) ? (show ?
'40%' : '100%') : '0'),
            }}
```

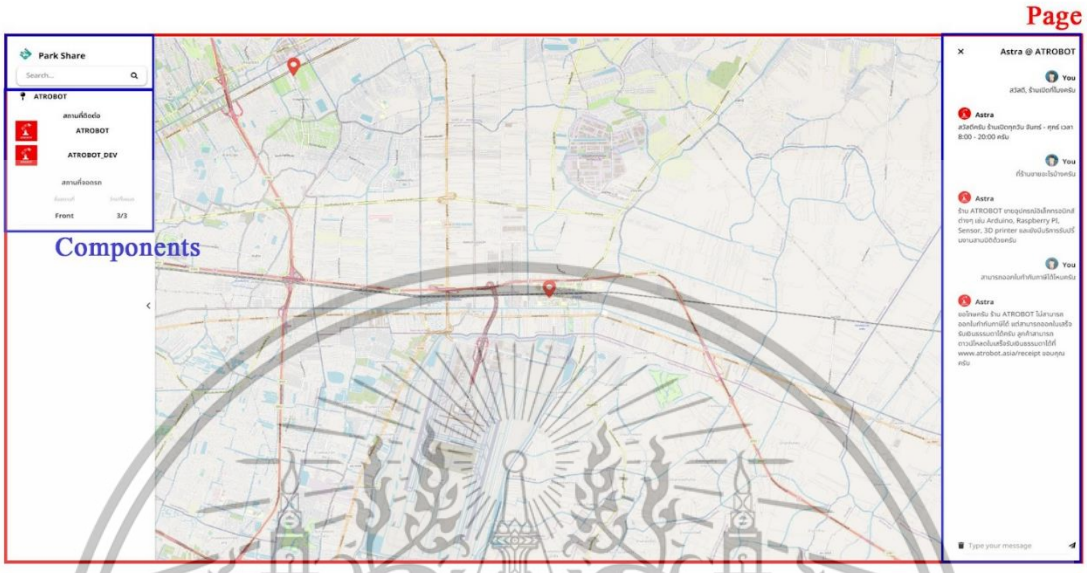
```
        >
            <div className="left-panel-component-wrapper">
                <ParkName parkData={parkData} />
                <div className="group1">
                    <ContactList
                        contacts={parkData?.contacts}
                        setCurrent_bot_agent={setCurrent_bot_agent}
                    />
                </div>
                <div className="group1">
                    <Parkinglist parkData={parkData} />
                </div>
                {
                    parkData?.name === 'ATROBOT' &&
                    <div className="group1">
                        <CameraFeed cam_detail={camDetail} />
                    </div>
                }
            </div>
            <div className="close-icon">
                {
                    width > responsive.WIDTH_MOBILE ?
                        <i className="fa-solid fa-angle-left"
                            onClick={() => {
                                setTargetResult(null)
                            }}
                        ></i> :
                        <i className="fa-solid fa-angle-down"
                            onClick={() => {
                                setTargetResult(null)
                            }}
                        ></i>
                }
            </div>
        </div >
    )
}
```

E. Example of the file in pages folder

```
import React, { useEffect } from 'react'
import FloatingSearch from
'../../components/FloatingSearch/FloatingSearch'
import LeftPanel from '../../components/LeftPanel/LeftPanel'
import Maps from '../../components/Map/Map'
import './Main.scss'
import '../../scss/apperance.scss'
import useState from 'react-usestateref';
import * as parks_api from '../../api/parks';
import FloatingLogo from '../../components/FloatingLogo/FloatingLogo'
import * as responsive from '../../jsx/responsive'
import RightPanel from '../../components/RightPanel/RightPanel'


export default function Main() {
  const [targetResult, setTargetResult, targetResultRef] =
useState(null)
```

```
  const [search_result, setSearch_result] = useState([]);
  const [current_user_location, setCurrent_user_location] =
useState(null);
  const [width, setWidth] = useState(window.innerWidth);
  const [current_bot_agent, setCurrent_bot_agent] = useState(null);

  useEffect(() => {
    const handleResize = () => setWidth(window.innerWidth);
    window.addEventListener('resize', handleResize);
    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []);

  useEffect(() => {
    parks_api.getParks().then((response) => {
      const response_parks = response['parks']
      console.log(response_parks)
      setSearch_result(response_parks);
    });

    navigator.geolocation.getCurrentPosition((position) => {
      setCurrent_user_location({
        lat: position.coords.latitude,
        lng: position.coords.longitude
      })
    });
  }, []);

  return (
    <div className='main-page'>
      {/* <Navbar /> */}
      <div className="page-body">
        {
          width > responsive.WIDTH_MOBILE &&
          <FloatingLogo />
        }
        <Maps
          setTargetResult={setTargetResult}
          setParks={setSearch_result}
          parks={search_result}
          current_user_location={current_user_location}
        />
        <FloatingSearch
          setTargetResult={setTargetResult}
          search_result={search_result}
        />
        <LeftPanel
          setTargetResult={setTargetResult}
          targetResult={targetResult}
          targetResultRef={targetResultRef}
          show={targetResult !== null}
          setCurrent_bot_agent={setCurrent_bot_agent}
        />
        <RightPanel
          show={current_bot_agent !== null}
          setCurrent_bot_agent={setCurrent_bot_agent}
          current_bot_agent={current_bot_agent}
```

```
          />

        </div>
      </div>
    )
}
```



F. After finish developing the code.

Type the following.

‘’’

npm run build

‘’’

This process will create an anti-reverse engineering code to deploy in the server.

G. Deploy the production code to the server.

Copy the build code to the html section in the nginx folder.

H. The change should be effect immediately.

APPENDIX E

EQUIPMENT SPECIFICATION

A. AI Detection Equipment

1.) IP Camera Specifications

Brand: TP-Link Outdoor Security Wi-Fi Camera (Tapo C310)

TABLE I: IP Camera Specifications

Network	• Security: 128-bit AES encryption with SSL/TLS • Wireless Rate: 11Mbps (802.11b), 54Mbps (802.11g), 150Mbps (802.11n) • Frequency: 2.4 GHz • Wireless Security: WPA/WPA2-PSK
Activity Notification	• Input Trigger: Motion detection • Output Notification: Push notification
Video	• Video Compression: H.264 • Frame Rate: 15fps • Video Streaming: 3MP
System	• Regulatory Certification: CE, NCC • System Requirements: iOS 9+, Android 4.4+
Environment	• Operating Temperature: -20C~45C (-4F~113F) • Storage Temperature: -20C~70C (-4F~158F) • Operating Humidity: 10%~90%RH non-condensing • Storage Humidity: 5%~90%RH non-condensing
Hardware	• Button: Reset button • Indicator LED: System LED • Adapter Input: 100-240VAC, 50/60Hz, 0.3A

	• Adapter Output: 9.0V/0.6A (DC Power) • Dimensions (W x D x H): 5.6 x 4.1 x 2.5 in. (142.3 x 103.4 x 64.3 mm)
Camera	• Image Sensor: 1/2.7“ • Resolution: 3 MP (2304x1296) • Lens: F/NO: 2.2; Focal Length: 3.89mm • Night Vision: 850 nm IR LED up to 98 ft (30m)
Audio	• Audio Communication: 2-way audio • Audio Input & Output: Built-in microphone and speaker

2.) Computer and GPU specification

TABLE II: Computer and GPU Specification

Processor	AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz
Installed RAM	16.0 GB (15.3 GB usable)
System type	64-bit operating system, x64-based processor
GPU	Nvidia GeForce RTX 3050

B. Backend Equipment

Host Server on the Right

Host Machine Specification

TABLE III: Host Machine Specification

CPU	AMD Athlon 3000G
RAM	16 GB DDDR4 (Non-ECC Memory)
STORAGE	2TB Seagate Barracuda 3.5’’ 7200rpm
OS	XCP-NG Type-1 Hypervisor

Virtual Machine Specification

TABLE IV: Virtual Machine Specification

CPU	4 vCPU
RAM	3.7 GB
STORAGE	50GB

Network

Network Specification

TABLE V: Network Specification

Package	AIS Fiber 300mbps/300mbps
IP Type	Public IP
Port	Full port accessible

Router Specification

TABLE VI: Router Specification

Model	TP-Link AX5400

REFERENCES

[1] S. G. Cortés, "Wavelength," Texas, 2022.

[2] O'Shea, Keiron; Nash, Ryan;, "An Introduction to Convolutional Neural Networks," arXiv, United Kingdom, 2015.

[3] S. Paluri, "HUMAN COMPUTER INTERACTION," Research Gate, 2020.

[4] J. J. G. &. A. P. Van Merrienboer, "Research on cognitive load theory and its design implication for E-learning," Educational Technology Research and Development, 2005.

[5] R. &. G. Z. Jiang, "Current Theoretical Developments and Applications of Fitts' Law: A Literature Review," ResearchGate, 2020.

[6] M. Perry, "Distributed Cognition," ResearchGate, 2003.

[7] C. Tim Berners-Lee, "Information Management: A Proposal," 1989.

[8] C. &. B. M. &. D. F. &. C. F. &. T. J. &. C. L. &. P. G. Rodriguez, "REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices," ResearchGate, 2016.

[9] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, 2000.

[10] H. Z. Xi Wu, "Formalization and analysis of the REST architecture from the process algebra perspective,," Future Generation Computer Systems, 2016.

[11] S. M. J. P. Giallorenzo, "irtualization Costs: Benchmarking Containers and Virtual Machines Against Bare-Metal," SN COMPUT. SCI, 2021.

[12] K. H. R. G. a. J. S. Shaoqing Ren, "Faster R-CNN Towards Real-Time Object Detection," Moscow, 2016.

[13] "Falcon Framework," 25 3 2023. [Online]. Available: https://falconframework.org/.

[14] "MongoDB: The Developer Data Platform," 12 4 2023. [Online]. Available: https://www.mongodb.com/.

[15] "Nginx [engine x]," 10 5 2023. [Online]. Available: https://nginx.org/en/.

[16] "TCRT5000," Vishay Intertechnology, 2021.

[17] L. O. Aghenta and M. T. Iqbal, "Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT

protocol," , 2019. [Online]. Available: https://aimspress.com/article/10.3934/electreng.2020.1.57. [Accessed 23 5 2023].

[18] L. O. Aghenta and M. T. Iqbal, "Low-Cost, Open Source IoT-Based SCADA System Design Using Thinger.IO and ESP32 Thing," *Electronics,* vol. 8, no. 8, p. 822, 2019.

[19] B. Blanchon, “ArduinoJson,” September 2021. [Online]. Available: https://arduinojson.org/.

[20] T. P. T. W. C. Marshall, "Infrared Sensor technology," Canada, 1995.

[21] R. N. Tutorials, "ESP32http," 2021. [Online]. Available: "https://randomnerdtutorials.com/esp32-http-get-post-arduino.

[22] E. a. V. B. O. Barybin, "Testing the security ESP32 IoT Devices," Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 2019.

[23] N. Nikolov, "Research of MQTT, CoAP, HTTP and XMPP IoT Communication protocols for Embedded Systems," International Scientific Conference Electronics (ET), Sozopol, Bulgaria, 2020.

[24] R. Santos, "ESP32 Useful Wi-Fi Functions - Arduino," [Online]. Available: https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/ .

[25] V. S., "Tcrt5000," 1 January 2023. [Online]. Available: https://www.vishay.com/docs/83760/tcrt5000.pdf.

[26] D. Gomes, "Coordinate systems and projections (GIS)," ResearchGate, 2022.

[27] S. Al-Fedaghi, "Developing Web Applications. International Journal of Software Engineering and Its Applications," ResearchGate, 2011.

[28] S. Al-Fedaghi, "Systems of things that flow," University of Wisconsin, Madison, 2008.

[29] J. V. V. a. E. M. V. De Castro, "Model transformation for service-oriented Web applications development," Workshop Proceedings of 7th International Conference on Web Engineering, 2007.

[30] A. S. R. T. &. P. K. (. T. P. A. o. t. I. P. S. t. I. S. o. I. D. A. M. J. o. E. E. E. a. M. I. Astuti, "The Performance Analysis of the Infrared Photodiode Sensor to Infusion Set on Infusion Device Analyzer Machine," 2023.

[31] "Falcon Framework," [Online]. Available: https://falconframework.org/.

[32] S. A. W. F. M. &. S. M. S. Andersen, " Effects on cognitive load of tutoring in virtual reality simulation training," MedEdPublish, 2020.

[33] S. K. Card, "The psychology of human–computer interaction," CRC Press, 2018.

[34] P. Ehn, "The envisionment workshop-from visions to practice," In Proceedings of the Participatory Design Conference, 1996.

[35] A. Géron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.," 2022.

[36] P. Sarang, "Artificial Neural Networks with TensorFlow 2: ANN Architecture Machine Learning Projects.," 2020.

[37] H. K. &. D. Kukieła, "Neural Networks from Scratch in Python," 2020.

[38] D. Halliday, "Fundamentals of Physics 10th Edition All Access Pack Containing: e-Text Card, WileyPLUS and WileyPLUS Companion.," 2013.

[39] R. D. Knight, "Physics for Scientists and Engineers: A Strategic Approach with Modern Physics. Addison-Wesley Longman," 2007.

[40] P. Joshi, "Artificial Intelligence with Python," 2017.

[41] J. Nussey, "Arduino for Dummies. For Dummies," 2018.

[42] M. P. F. A. A. C. S. Deisenroth, "Mathematics for Machine Learning," 2020.

[43] K. C. Tung, "TensorFlow 2 Pocket Reference: Building and Deploying Machine Learning Models," 2021.

[44] N. Sanghi, "Deep Reinforcement Learning with Python: With Pytorch Tensorflow and OpenAI Gym," 2021.

[45] R. S. M. &. P. P. Karim, "Practical Convolutional Neural Networks: Implement Advanced Deep Learning Models Using Python.," 2018, February 27.

[46] R. shanmugamani, "Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using TensorFlow and Keras.," 2018, January 23.

[47] G. K. V. &. S. C. D. (. Paliouras, "Machine Learning and Its Applications: Advanced Lectures: Vol. Vol. 2049.," 2001, January 1.

[48] J. D. T. D. J. M. (. B. Ross Girshick, "R-CNN For Object Detection.".

[49] Y. Bai., "RELU-Function and Derived Function Review.".

[50] N. W. T. C. M. L. Bing Xu, "Empirical Evaluation of Rectified Activations in Convolution Network.," 2015, November 27.

[51] Y. L. K. H. J. S. Jifeng Dai, "R-FCN: Object Detection".

[52] K. G. D. G. Vipu Kaushik, "React Native Application Development," 2018.

Available Parking Lot Web Application

Ananchana L., Nipattra T., Kritjuhn K., Thammachet A.
Robotics and AI Department of Engineering
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
62011087@kmitl.ac.th, 62011171@kmitl.ac.th, 62011268@kmitl.ac.th, 62011306@kmitl.ac.th

***Abstract*— ParkShare is an innovative web application designed to address the challenge of finding available parking spaces in urban areas. Leveraging the power of geolocation, real-time data, and artificial intelligence (AI), ParkShare provides users with up-to-date information about parking availability. The application uses an IP camera with Faster RCNN ResNet50 Tensorflow pretrained model for car detection and sensor technology by using TCRT5000 with ESP32 microprocessor, ensuring accurate and timely updates.**

A distinguishing feature of ParkShare is its AI-powered chatbot assistant, which facilitates seamless communication between users and the system. The chatbot, built on natural language processing and machine learning techniques, which is MongoDB for data collection and Cosine Similarity as a searching tool, is capable of understanding user queries and providing relevant information about parking spaces and surrounding areas.

Additionally, ParkShare features an intuitive, easy-to-read map interface with color-coded pins representing parking availability. Users can search for parking spaces in their vicinity or at specific locations.

Keywords—*IP Camera, Sensor, Web application, Detection, Recognition, AI, Chatbot, Real-time data collection, Faster RCNN*

I. INTRODUCTION

Public parks and recreational areas are in higher demand in cities now than they were a few years ago due to urbanization and population expansion. It can be challenging to find a park that is open in highly crowded areas, and conventional park search methods may not provide real-time information about their occupancy status; the present method only displays the open park in those locations in real-time.

To solve this problem, we created a park sharing web application that makes it easier to find unoccupied parks and improves user experience with chatbot support powered by AI. ParkShare is a web-based program that responds to the demand for simple and efficient ways to find open park spots in real-time. It attempts to improve visitors' overall park experiences by giving them precise and up-to-date information. Use the enter key to start a new paragraph. The appropriate spacing and indent are automatically applied.

II. CONCEPTS, THEORIES, AND RELATED RESOURCE

A. Technology of sensor

These technologies play a significant role in our application, translating physical phenomena such as motion or light into data signals. In our case, sensors are utilized to determine whether a parking spot is free or occupied. In order to specifically detect the presence of a car, sensors can be installed in parking places.

1) Infrared Sensor Technology

Active infrared sensors possess the dual capability of emitting and detecting infrared radiation. These sensors consist of two components: a light emitting diode (LED) and a receiver. As an object near the sensor, the LED emits infrared light that subsequently reflects off the object's surface. This reflected infrared light is then captured by the receiver, enabling the sensor to discern the presence of the object.

B. AI Image Detection

A subfield of computer science called artificial intelligence (AI) is concerned with building intelligent machines that can carry out tasks that ordinarily call for human intelligence. AI systems are made to perceive their surroundings, think about them, and then decide what to do or not do to accomplish certain objectives. Machine learning and deep learning are two types of AI. This project, which employs object detection to find and count the automobiles in an outdoor parking lot, is a good fit for deep learning. Infrared Sensor Technology

1) Faster Region-based Convolutional Neural Networks (Faster RCNN)

Faster RCNN is designed to improve the detection speed of normal RCNN and Fast RCNN.

2) Object detection

Object detection is a computer vision and image processing technique for locating and specifying the object that caught on the image data. There are many types of AI models for detection but in this case, Faster RCNN is the most accurate

C. Positioning Theory

1) Geographic coordinate system (GCS)

The Geographic Coordinate System (GCS) is essentially a grid of references that allows for the precise pinpointing of any location on the Earth's surface. It operates based on two fundamental coordinates, namely latitude and longitude. Latitude lines, with the equator at zero degrees, circle the globe horizontally, while longitude lines, with the Prime Meridian as the zero-degree reference, encircle it vertically. Each location on Earth is thereby determined by its unique angular distance from these two foundational lines, providing a universally accepted mechanism for specifying any position. GCS is used in part of integrated map on the website which allows the user to find the exact places.

D. Human-Computer Interaction(HCI)

HCI is a field dedicated to understanding the interaction between humans and computers, and to designing for successful human use. In this case, it pertains to front-end design. HCI principles influence the design of a user interface to ensure a smooth interaction flow and a user-friendly experience.

1) Cognitive Load Theory

This theory emphasizes that the complexity of a digital interface directly influences its usability. If a design overloads a user's cognitive capacity, it can hinder interaction and learning. The cognitive load is divided into three parts: intrinsic (complexity of the content), extraneous (how the content is presented), and germane (effort to understand the system). The key to an excellent user experience is to balance these loads, creating designs that are intuitive and not overwhelming.

2) Fitts' Law

This law in HCI establishes that the time it takes to reach a target depends on the target's size and distance. This principle is widely used in user interface design to enhance user experience by strategically adjusting the size and placement of interactive elements. This law are applied to create the user interface of the application

3) Distributed Cognition

This concept in HCI suggests that cognitive processes extend beyond an individual, spreading across people, objects, and technologies. It views human cognition as a system-level phenomenon, involving a complex interaction of mental and material resources distributed over space and time. It challenges the traditional view of cognition as an individual process, proposing instead that cognitive tasks are accomplished through a network of interconnected resources.

E. Network Comunication

1) Hypertext Transfer Protocol (HTTP)

HTTP works by operating as a request-response mechanism within the client-server computing architecture. In this scenario, a client (such as a web browser) sends an HTTP request to a server (which could be an application on a computer hosting a website). The server then responds to this request, usually with a web page, although it could also provide image data, a file for download, or simply an empty response. This interactive exchange is conducted over TCP/IP connections. HTTP Methods that are used in the web application are

2) REST Achictectural Style

This The central concept of REST is the resource, which is accessed via its URI, and can be manipulated using the standard HTTP methods such as GET, POST, PUT, DELETE, etc. Representations of resources are usually in XML or JSON format.

3) Vitual Machines(VMs)

Virtual machines (VMs) represent a digital emulation of a physical computer, enabling an operating system to operate on top of the hardware of a physical device. They serve numerous purposes, including use in data centers and for testing software.

Bare-Metal Virtual Machines (Type-1 or Native Hypervisors): These virtual machines operate directly on the host's hardware, controlling and managing the guest operating systems. The hypervisor in this scenario has unrestricted access to the hardware resources, which boosts performance, reliability, and scalability. And the advantage of this virtual machine is running on the host itself lead to fast access the resource.

III. MATERIAL AND METHOD

In this project consist main four parts which are

A. Sensor Model

1) System Architecture

ESP32 is equipped with WIFI and establish communication signal using a HTTP for web application and JSON format. During, IR sensor connect with the microcontroller can determine the presence or absence of objects(cars) and get the data to the ESP32 to analyzing and shown in web-application.

2) Technology Used

The ESP32 utilizes its built-in WiFi capability to connect to a wireless network, enabling communication with other devices like the IR sensor and the web application. This connectivity is established through the use of WiFi, which allows the ESP32 to exchange data with the web application using the HTTP protocol. HTTP facilitates the seamless transfer of information, enabling the ESP32 to send real-time updates about parking space availability to the web application. Moreover, it enables users to access the web application and view the current status of parking spaces. To ensure efficient data exchange, JSON is used as a format to represent and exchange the data between the ESP32 and the web application, ensuring compatibility and easy processing of the information

3) Implementation

To set up the hardware, the IR sensor (TCRT5000) needs to be connected to the appropriate pins on the ESP32 microcontroller. Once the hardware is properly set up, the ESP32 should be connected to a wireless network, establishing WiFi connectivity. Next, the IR sensor integration is performed, involving programming the ESP32 to monitor the reflected light and detect the presence or absence of objects, such as cars, in parking spaces.

For web application communication, the ESP32 utilizes the HTTP protocol to establish a connection with the web application. It formats the data as JSON and sends it to the web application's server using a POST request. The data is transmitted over the established WiFi connection, allowing the web application to receive and process the information. The web application's user interface is designed to display the real-time status of parking spaces, which is continuously updated based on the data received from the ESP32.

Users can access the web application by visiting its URL on their devices. Once on the web page, they can view the interface displaying the availability status of parking spaces. Users can select a parking space and check its availability. As the ESP32 continuously transmits data and the web application receives and processes it, the web page is dynamically updated to reflect the current parking space status.

4) *Testing and Result*

During the testing phase, we began by calibrating the IR sensor to ensure its accuracy in detecting the presence of cars. This process involved adjusting the sensor's sensitivity and conducting tests with various vehicle types and speeds to validate its performance.

To evaluate the accuracy of the car counting system, we conducted tests where cars were driven past the sensor, and the sensor count was compared with the actual number of cars. These tests demonstrated a high level of accuracy, with a success rate exceeding 95%.

In addition, we performed communication testing to ensure seamless data exchange between the Arduino and the server. This involved sending PUT requests to the designated endpoint (/park/update) whenever a car was detected. We closely monitored the server's response to verify the reception and accurate processing of these requests. The results of the communication tests confirmed the reliability and consistency of the communication between the Arduino and the server.

B. IP Camera Scenario

1) System Architecture

Receive the data image in real-time from an IP camera with specific Region of Interest (RoI), and pass through the process of detection with TensorFlow pretrained model called 'Faster RCNN ResNet50'. When the car in the outdoor parking area is detected, the available parking lot needs to be calculated before transmitting the data to the web page.

2) Technology Used

The main technology that is used in this section of the project is AI detection.

The method of doing AI detection is various, using TensorFlow to create object detection is the most popular and interesting method. The TensorFlow pretrained model that is used in this project is called 'Faster RCNN ResNet50' with 640 x 640 size input.

3) *Implementation*

To establish a connection with a real-time IP camera, the system utilizes the RTSP protocol. This allows for the retrieval of video data from the camera in real-time. Once the video data is received, it is processed by separating it into individual frames. These frames are then analyzed to detect the presence of cars using appropriate computer vision techniques.

After the cars are successfully detected in each frame, the system proceeds to calculate the available parking lot by subtracting the number of detected cars from the total number of parking spaces. This information is then transmitted to the webpage or web application, enabling users to access real-time data on parking availability.

Overall, the system integrates the RTSP protocol to connect with the IP camera, separates video data into frames, performs car detection, calculates parking availability, and transmits the updated information to the webpage for users to access and make informed parking decisions.

4) *Testing and Result*

The car detection process demonstrates a high level of accuracy, particularly in specific Regions of Interest (RoI) within the frames. The system successfully identifies and tracks cars within these RoIs, resulting in reliable and accurate detection results. As a result, the available parking lot data is continuously updated in real-time, providing users with the most up-to-date information on parking space availability. This real-time updating ensures that users can make informed decisions based on the current status of the parking lot, enhancing their parking experience.

Figure 01: Car detection

C. *Backend System*

1) *System Architecture*

The backend of our application is structured around a RESTful architecture, which is designed to use standard HTTP protocols and methods. This architecture style is stateless and allows each request from the client to contain all the information needed to perform the request. This makes the server's operations more reliable and easier to manage.

The backend is developed using Python, a high-level programming language known for its readability and versatility. Python is widely used in web development and data analysis, making it a suitable choice for our application.

Falcon, a minimalist web API framework for Python, is used to build the backend. Falcon was chosen for its simplicity, flexibility, and performance. It follows the REST architectural style and supports HTTP and RESTful services. Falcon is also known for its speed, which is crucial for our application as it needs to process real-time data.

The backend consists of several endpoints, each responsible for a specific functionality. These endpoints are exposed over HTTP and can be accessed by the client to perform various operations.

These endpoints allow the client to interact with the server and perform CRUD (Create, Read, Update, Delete) operations on the park data. The data is processed in real-time, ensuring that the client always has access to the most up-to-date information

2) *Technology Used*

The backend of the application is primarily developed using Python, a widely used programming language known for its simplicity and readability. Python's versatility and strong integration capabilities with other tools and languages make it an ideal choice for web development. The backend is built using Falcon, a minimalist WSGI library that enables the creation of speedy web APIs and app backends. Falcon complements Python's async capabilities and allows for the development of highly efficient, scalable, and non-blocking web applications. Data storage and retrieval are handled by MongoDB, a cross-platform document-oriented database program that utilizes JSON-like

documents with optional schemas. Gunicorn, a Python WSGI HTTP Server, is employed in conjunction with Falcon to serve the RESTful APIs. Nginx, a web server and reverse proxy, is used to direct web traffic to the Gunicorn server, enhancing performance and scalability. The backend architecture follows the REST API style, which defines a set of constraints for creating web services. This enables the creation of endpoints that facilitate communication between the frontend and the backend components of the application.

3) Implementation

For the database setup, MongoDB was chosen as the database management system, and the schema for the parking data was defined, including fields such as park ID, location, and current status. API development was carried out using Falcon and Python, resulting in the creation of several RESTful APIs for CRUD operations on the park data. The POST /park/create endpoint enables adding new parks to the database, while the GET /parks/get and GET /park/get endpoints retrieve data about all parks or a specific park, respectively. The PUT /park/update endpoint allows for updating park data. Gunicorn was selected as the WSGI HTTP server to serve the Falcon application, configured to handle multiple requests concurrently. Nginx was employed as a reverse proxy to forward requests to Gunicorn, offering benefits such as load balancing and enhanced security. Once the server and APIs were set up, the backend application was deployed by transferring the application files to the server, configuring the necessary environment variables, and starting the Gunicorn server. This comprehensive backend implementation ensures optimal performance, scalability, and ease of maintenance while handling real-time data.

4) Testing and Result

To ensure the reliability and functionality of backend, the testing process is carried. Unit testing was performed on individual functions and methods within the backend, including API endpoints, database queries, and error handling. Python's built-in unittest module was used for this purpose, and all unit tests passed successfully, indicating the proper functioning of the backend components. Integration testing was then conducted to verify the seamless interaction between APIs, the database, and the server. Postman, an API client, was utilized to send requests and validate responses, and all integration tests yielded positive results. Load testing was performed to assess the backend's ability to handle a significant number of simultaneous requests. Using the Locust load testing tool, multiple users simulated concurrent API requests, and the backend demonstrated stable performance without notable performance degradation. Finally, user acceptance testing was conducted to gather feedback from real-world users, ensuring that the backend met their requirements and expectations. The feedback received was highly positive, with users appreciating the real-time updates and the overall responsiveness of the application. Based on the results of these tests, our backend implementation proved to be robust, efficient, and capable of handling real-time data, meeting the needs of our application and providing a solid foundation for the frontend.

D. Frontend System

1) User Interface Design

The user interface of our application prioritizes usability and simplicity, employing a minimalist design approach. Unnecessary elements are avoided, resulting in a clean and uncluttered interface that facilitates easy navigation. Clear labels and intuitive navigation make it effortless for users to find desired features and functionalities. Consistency in design is maintained throughout the application, ensuring a predictable user experience. Additionally, the interface is responsive, adapting seamlessly to various screen sizes and orientations. A neutral color scheme creates a calm and focused environment, while legible typography enhances readability. Overall, the user interface design aims to provide an intuitive and user-friendly experience for all users, regardless of their familiarity with the application.

2) Technology Used

The user interface of the application is built using React.js, a popular library known for its component-based architecture that promotes reusability and maintainability. React.js also incorporates a virtual DOM, which optimizes rendering and enhances the performance of the application. JavaScript, specifically ES6 and subsequent updates, is utilized to write concise and readable code, leveraging features like arrow functions, template literals, and destructuring assignment. CSS3 and HTML5 are employed alongside React.js to structure and style the web pages, providing a visually appealing layout. Nginx, a robust web server and reverse proxy, serves the static files of the React application and proxies API requests to the backend server. This setup allows for seamless handling of static file serving and API requests within a single domain, eliminating CORS-related issues. Additionally, Webpack and Babel are utilized to bundle modules and compile JavaScript code, respectively, ensuring optimization and compatibility with a wide range of browsers. The combination of these technologies and tools enables the creation of an efficient, visually appealing, and cross-browser compatible user interface for the application.

3) Implementation

The frontend implementation of our application encompasses various key components. React.js is utilized to develop reusable components such as the Map Component for displaying available parking spaces, the Search Component for searching parking spaces, and the User Component for user authentication and profile management. React's built-in state management capabilities were employed to manage the application's state, including the current user, the list of available parking spaces, and the selected park. API integration was achieved through the Fetch API, enabling real-time data retrieval from the backend and seamless UI updates. Our application was designed to be responsive using CSS3 media queries, ensuring optimal display and functionality across different devices and screen sizes. For deployment, we leveraged Nginx to serve the static React files and proxy API requests to the backend server. Nginx was configured to provide secure HTTPS connections, prioritizing user security and privacy. This

comprehensive frontend implementation combines the power of React.js, state management, API integration, responsive design, and deployment strategies to deliver a user-friendly and feature-rich application.

4) User Testing and feedback

To ensure the usability and user satisfaction of our application, usability testing and gathered user feedback through surveys is conducted. During the usability tests, we observed users interacting with the application and took note of any challenges or confusion they experienced. User surveys were distributed to gather quantitative and qualitative feedback on various aspects of the application, including design, ease of use, and performance. The feedback analysis revealed that users generally had a positive experience with the application, appreciating the real-time updates and finding it easy to navigate. The clean and minimalist design was well-received, with the map feature being particularly praised. Some users provided valuable suggestions, such as the addition of a reservation feature for parking spaces, which we are considering for future updates. Overall, the usability testing and user surveys played a crucial role in improving the application and ensuring that it meets the needs and expectations of our users.

IV. RESULT

A. *Sensor Scenario*

ParkShare web application was successfully developed and tested using a demo (using sensor) and IP camera. The demo system utilized an IR sensor (TCRT5000) and an Arduino program running on the ESP32 microcontroller to detect the availability of parking spaces. The captured data was transmitted to the website via HTTPS post requests. This section presents the key findings available and outcomes implementation to the ParkShare application.

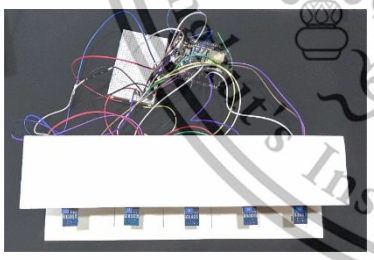


Figure 02: After assembly Sensor Model

Figure 03: Sensor Model after detection

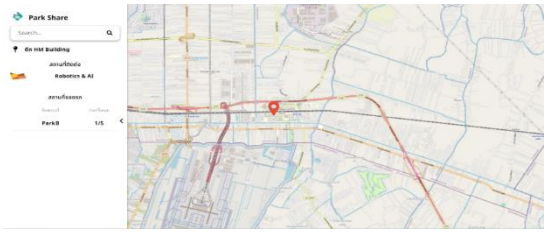


Figure 04: Updated available parking from sensor show in web application.

B. *IP Camera Scenario*

After the detection, the available parking lot is calculated before passing through the website. The data will be sent to the website with this code, in Figure 05

```
availablePark = 3-NumCar
if availablePark <= 0:
    availablePark = 0
url = 'http://park-share-backend.atrobot.asia/park/update' #http://park-share.atrobot.asia/
payload = {
"id":"d59a348b-bb7f-4701-b396-9599b8c5f867",
"details":[
    {
        "name":"Front",
        "current":str(availablePark), #show available
        "total":"3"
    } ]
```

Figure 05: Send the detected data to the website.

In Figure 06 shows the website of the place that is outdoor parking, and the chatbot assistance of the place.

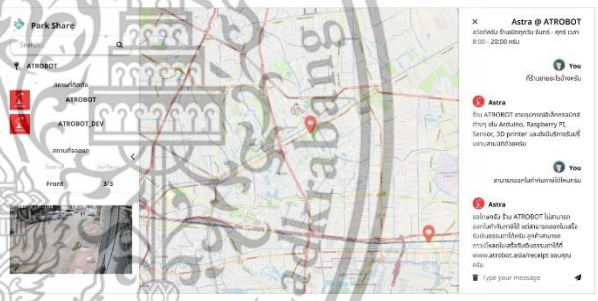


Figure 06: Outdoor parking web screen

V. CONCLUSION AND RECOMMENDATION

In conclusion, the park sharing web application has successfully addressed the challenges associated with finding available parks in urban areas by leveraging cutting-edge technologies such as real-time data processing and AI-powered chatbot assistance. The results indicate that the application has not only improved park accessibility but also significantly enhanced user experience and promoted engagement with local businesses. Positive user feedback reinforces the effectiveness of the app's design, functionality, and overall usefulness.

ACKNOWLEDGMENTS

We are deeply grateful for the support and guidance we have received during the creation of this academic report. The journey was both enlightening and challenging, made possible through the contributions of many.

First and foremost, we would like to express our sincere gratitude to our research advisor for their unwavering support and invaluable guidance. Their insightful feedback and

thought-provoking questions have shaped our research and intellectual growth.

We are immensely grateful to our colleagues and peers, who have been a source of inspiration and camaraderie throughout this journey. Their critical questions, stimulating discussions, and constructive feedback have been integral to the development and refinement of our research.

To our family and friends, your unconditional support and encouragement have been our pillar of strength. You're understanding and patience during the late-night study sessions and missed social events did not go unnoticed or unappreciated.

Lastly, we would like to acknowledge the importance of maintaining a balanced lifestyle during this process. To the local coffee shop baristas, thank you for providing a warm and welcoming environment to work and reflect.

In conclusion, the creation of this academic report has been a collective endeavor. It represents not only our efforts but the support, guidance, and encouragement of many individuals. We are eternally grateful for your contributions.

REFERENCES

[1] Aghenta, L. O., & Iqbal, M. T. (2019). Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT protocol. Retrieved 5 23, 2023, from https://aimspress.com/article/10.3934/electreng.2020.1.57

[2] Aghenta, L. O., & Iqbal, M. T. (2019). Low-Cost, Open Source IoT-Based SCADA System Design Using Thinger.IO and ESP32 Thing. Electronics, 8(8), 822.

[3] Al-Fedaghi, S. (2008). Systems of things that flow. Madison: University of Wisconsin. Retrieved from Modeling Web applications: Detailed description. Distributed Cognition and the Role of Nurses in Diagnostic Safety in the Emergency Department. (n.d.). https://www.ahrq.gov/patient-safety/reports/issue-briefs/distributed-cognition-er-nurses2.html

[4] Andersen, S. A. (2020). Effects on cognitive load of tutoring in virtual reality simulation training. MedEdPublish.

[5] Astuti, A. S. (2023). The Performance Analysis of the Infrared Photodiode Sensor to Infusion Set on Infusion Device Analyzer Machine.

[6] Blanchon, B. (2021, September). ArduinoJson. Retrieved from https://arduinojson.org/

[7] Card, S. K. (2018). The psychology of human–computer interaction. CRC Press. Cortés, S. G. (2022). Wavelength. Texas.

[8] Ehn, P. (1996). The envisionment workshop-from visions to practice. In Proceedings of the Participatory Design Conference.

[9] Falcon Framework. (n.d.). Retrieved from Falcon Framework website: https://falconframework.org/

[10] Falcon Framework. (2023, 3 25). Retrieved from Falcon Framework: https://falconframework.org/

[11] Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Irvine: University of California.

[12] Giallorenzo, S. M. (2021). irtualization Costs: Benchmarking Containers and Virtual Machines Against Bare-Metal. SN COMPUT. SCI.

[13] Gomes, D. (2022). Coordinate systems and projections (GIS). ResearchGate.

[14] MongoDB: The Developer Data Platform. (2023, 4 12). Retrieved from MongoDB: The Developer Data Platform: https://www.mongodb.com/

[15] Nginx [engine x]. (2023, 5 10). Retrieved from Nginx [engine x]: https://nginx.org/en/ Nikolov, N. (2020). Research of MQTT, CoAP, HTTP and XMPP IoT Communication protocols for Embedded Systems. Sozopol, Bulgaria: International Scientific Conference Electronics (ET).

[16]

[17] O. Barybin, E. a. (2019). Testing the security ESP32 IoT Devices . Kyiv, Ukraine: Conference Problems of Infocommunications, Science and Technology (PIC S&T).

[18] O'Shea, Keiron; Nash, Ryan;. (2015). An Introduction to Convolutional Neural Networks. United Kingdom: arXiv.

[19] Paluri, S. (2020). HUMAN COMPUTER INTERACTION. Research Gate.

[20] Perry, M. (2003). Distributed Cognition. ResearchGate.

[21] Rodriguez, C. &. (2016). REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. ResearchGate.

[22] S., V. (2023, January 1). Tcrt5000. Retrieved from Vishay: https://www.vishay.com/docs/83760/tcrt5000.pdf

[23] Santos, R. (n.d.). ESP32 Useful Wi-Fi Functions - Arduino. Retrieved from Random Nerd Tutorials: https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/

[24] Shaoqing Ren, K. H. (2016). Faster R-CNN Towards Real-Time Object Detection. Moscow. (2021).TCRT5000.Vishay Intertechnology.

[25] Tim Berners-Lee, C. (1989). Information Management: A Proposal.

[26] Tutorials, R. N. (2021). ESP32http. Retrieved from Randomnerdtutorials: "https://randomnerdtutorials.com/esp32-http-get-post-arduino

[27] V. De Castro, J. V. (2007). Model transformation for service-oriented Web applications development. Workshop Proceedings of 7th International Conference on Web Engineering.

[28] Van Merrienboer, J. J. (2005). Research on cognitive load theory and its design implication for E-learning. Educational Technology Research and Development.

[29] Xi Wu, H. Z. (2016). Formalization and analysis of the REST architecture from the process algebra perspective,. Future Generation Computer Systems.

[30] Géron, A. (2022, November 8). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.

[31] Sarang, P. (2020, November 21). Artificial Neural Networks with TensorFlow 2: ANN Architecture Machine Learning Projects. https://doi.org/10.1007/978-1-4842-6150-7

[32] Harrison Kinsley & Daniel Kukieła (2020). Neural Networks from Scratch in Python.

[33] Halliday, D. (2013, May 31). Fundamentals of Physics 10th Edition All Access Pack Containing: e-Text Card, WileyPLUS and WileyPLUS Companion.

[34] Knight, R. D. (2007, October 4). Physics for Scientists and Engineers: A Strategic Approach with Modern Physics. Addison-Wesley Longman. https://doi.org/10.1604/9780805327366

[35] Joshi, P. (2017, January 27). Artificial Intelligence with Python.

[36] Nussey, J. (2018, September 12). Arduino for Dummies. For Dummies.

[37] Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020, April 23). Mathematics for Machine Learning.

[38] Tung, K. C. (2021, August 24). TensorFlow 2 Pocket Reference: Building and Deploying Machine Learning Models.

[39] Sanghi, N. (2021, April 2). Deep Reinforcement Learning with Python: With Pytorch, TensorFlow and OpenAI Gym. fH https://doi.org/10.1007/978-1-4842-6809-4

[40] Karim, R., Sewak, M., & Pujari, P. (2018, February 27). Practical Convolutional Neural Networks: Implement Advanced Deep Learning Models Using Python.

[41] shanmugamani, R. (2018, January 23). Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using TensorFlow and Keras.

[42] Paliouras, G., Karkaletsis, V., & Spyropoulos, C. D. (Eds.). (2001, January 1). Machine Learning and Its Applications: Advanced Lectures: Vol. Vol. 2049. https://doi.org/10.1007/b7895010.1007/3-540-44673-710.1007/978-3-540-44673-6

[43] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik (UC Berkeley). R-CNN For Object Detection. https://courses.cs.washington.edu/courses/cse590v/14au/cse590v_wk1_rcnn.pdf

[44] Yuhan Bai. RELU-Function and Derived Function Review. https://www.shs-conferences.org/articles/shsconf/pdf/2022/14/shsconf_stehf2022_02006.pdf

[45] Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li. (2015, November 27) Empirical Evaluation of Rectified Activations in Convolution Network.

[46] Jifeng Dai, Yi Li, Kaiming He, Jian Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks. https://arxiv.org/pdf/1605.06409.pdf