ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Sistemas e Informática

Tema: Sistema integrador de información en tiempo real, para la gestión y control de combustible, empleando tecnología IoT implementada a controladores industriales, en la estación de servicio “Eco el Oasis”.

Autores:

Ayala Santamaría David Sebastián

Ruiz Borja Mauricio Fernando

Director del Proyecto:

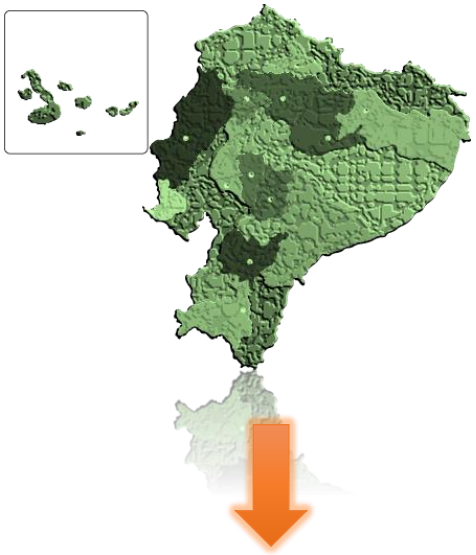
PhD. Ramiro Nanac Delgado Rodriguez

1

Agenda

- Antecedentes
- Planteamiento del Problema
- Objetivos
- Alcance
- Metodología y Modelo IoT
- Desarrollo del Prototipo
- Validación del Prototipo
- Resultados
- Conclusiones

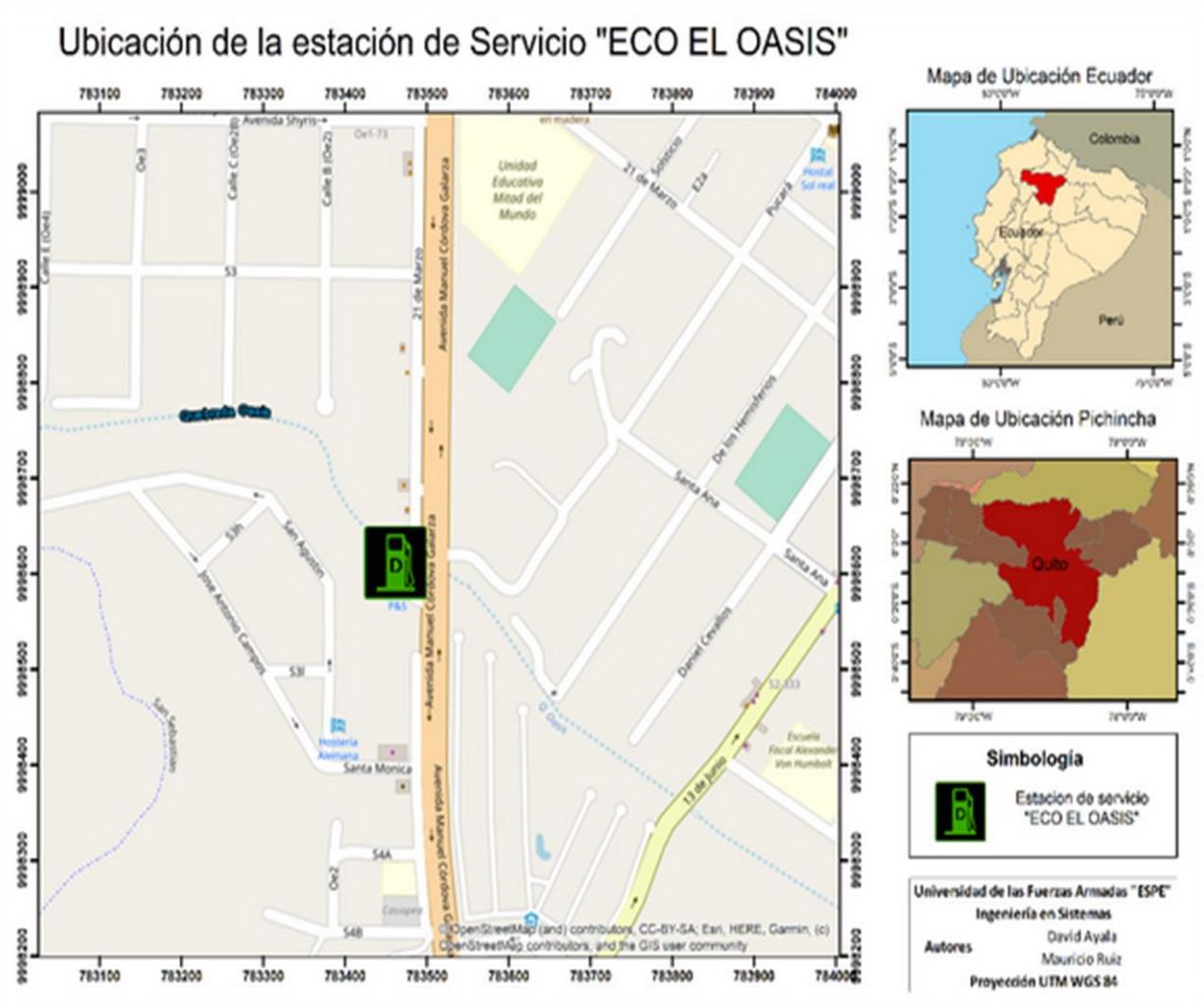
Antecedentes



- Distribución y Comercialización de Combustibles.
- Desarrollo Económico del país.
- Infraestructura Tecnológica de punta.
- Procesos Interconectados para la calidad del servicio.

3

Antecedentes



Transformación Tecnológica y desarrollo de nuevos sistemas.

Agilidad para la interacción de variable y elementos.

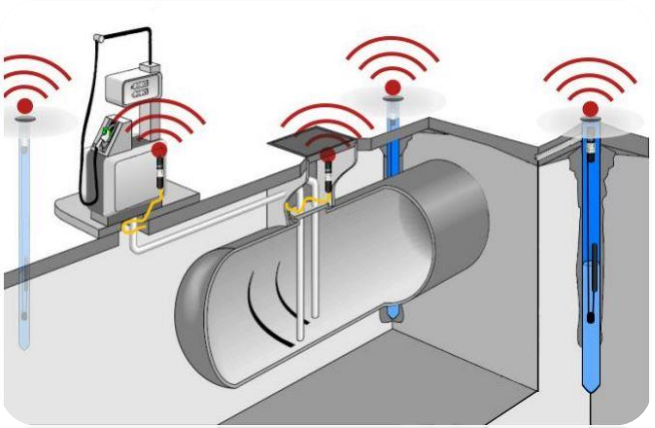
Propuestas de infraestructura tecnológica de bajo costo.

Planteamiento del Problema

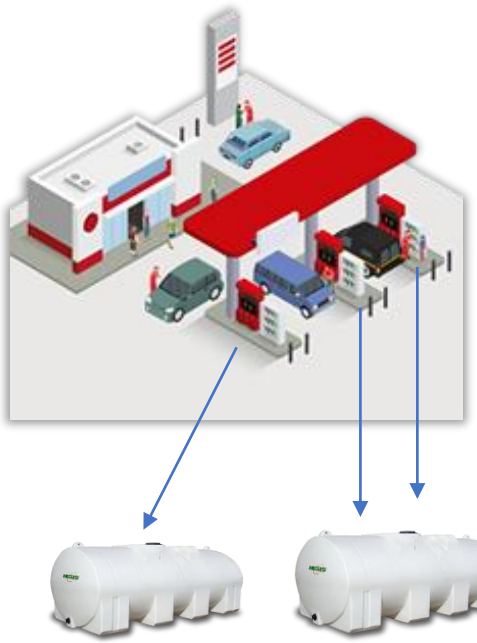
El petróleo en relación a la economía y la sociedad.

Industrias y organizaciones haciendo uso del petróleo.

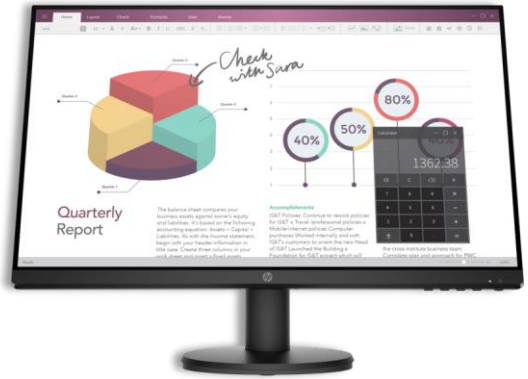
Fallos en la infraestructura tecnológica y tecnificación de procesos.



Planteamiento del Problema



Transactor



Servidor y Monitor

Surtidor y Tanques

Objetivos

OBJETIVO GENERAL

Desarrollar un sistema integrador de información en tiempo real, entre un controlador industrial (Transactor) y un Raspberry PI, basado en software, para la gestión y control de combustible optimizando el costo de la infraestructura tecnológica, mediante la utilización de tecnologías IoT por medio del lenguaje de programación PYTHON, dentro de la estación de servicio “ECO EL OASIS”.

OBJETIVOS ESPECÍFICOS

01

- Realizar un análisis comparativo, de diseños y modelos de referencia para arquitecturas IoT (Internet Of Things), con el fin de seleccionar un modelo que solvente las necesidades para la integración entre un controlador industrial (Transactor) y un Raspberry PI..

02

- Establecer un sistema de comunicación, que envíe y reciba información en tiempo real, mediante protocolos definidos entre un controlador industrial (Transactor) y un Raspberry PI.

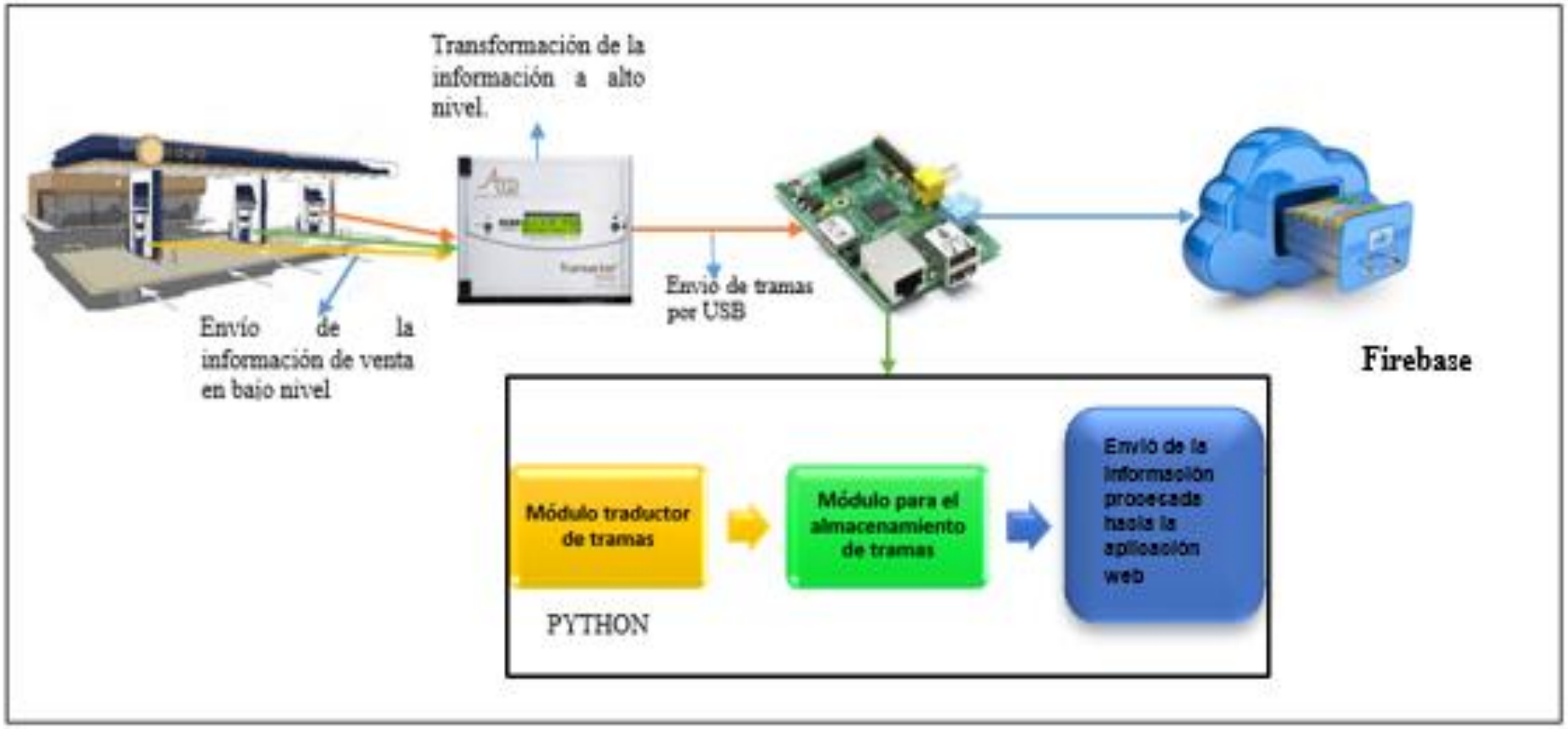
03

- Gestionar las tramas generadas desde un controlador industrial (Transactor) en un Raspberry Pi, con el fin de almacenar las mismas, en una base de datos NoSQL y mediante un aplicativo web poder visualizarlas. .

04

- Procesar la información obtenida por la comunicación establecida mediante un modelo de arquitectura IoT (Internet Of Things) y presentarla en una aplicación, desarrollada en un lenguaje de propósito general PYTHON.

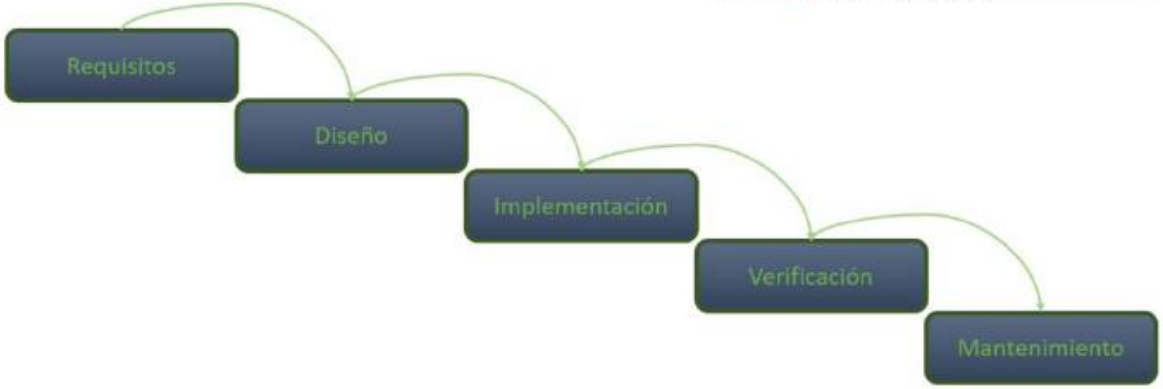
Alcance

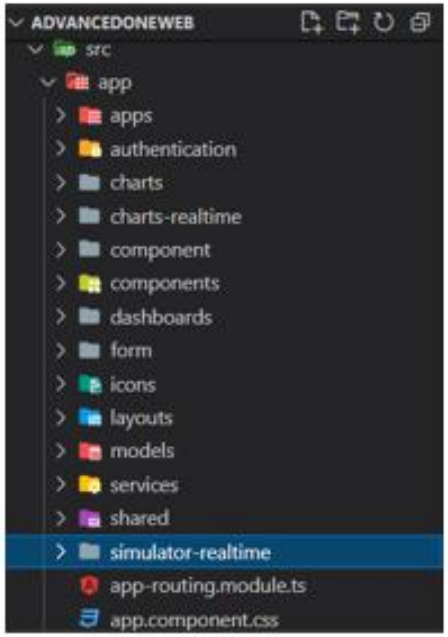

Transformación de la información a alto nivel.
Envío de la información de venta en bajo nivel
Envío de tramas por USB
Firebase
Módulo traductor de tramas
Módulo para el almacenamiento de tramas
Envío de la información procesada hasta la aplicación web
PYTHON

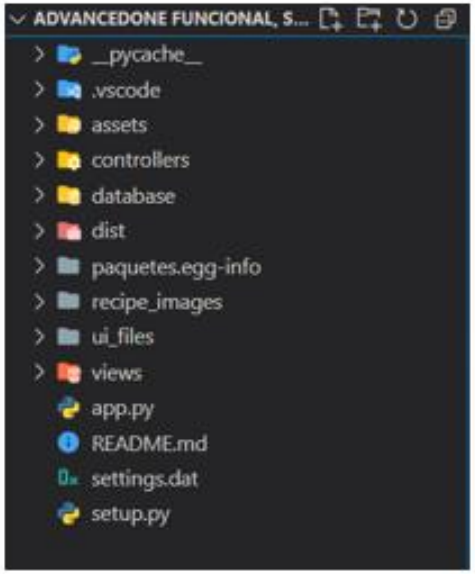
Metodología

Método Científico

Modelo en Cascada







Patrón MVC

Modelo IoT

Análisis de artículos científicos

- **Preguntas de investigación**
  - ¿Existen modelos de arquitectura IoT bien definidos que puedan interactuar simultáneamente con un desarrollo de software enfocado en conceptos de tiempo real?
  - ¿Cómo se gestiona el intercambio de información masivo entre dispositivos en un modelo de arquitectura IoT?
- **Selección de artículos**
- **Construcción de cadena de búsqueda**
- **Conclusión**
- **Análisis comparativo entre modelos**

Modelo IoT

7 Capas

Capa 1: Cosas

Capa 2: Conectividad

Capa 3: Infraestructura

Capa 4: Información

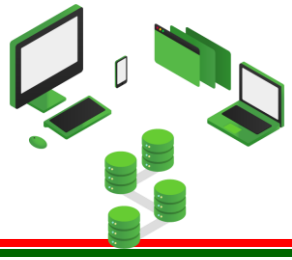
Capa 5: Análisis

Capa 6: Aplicación

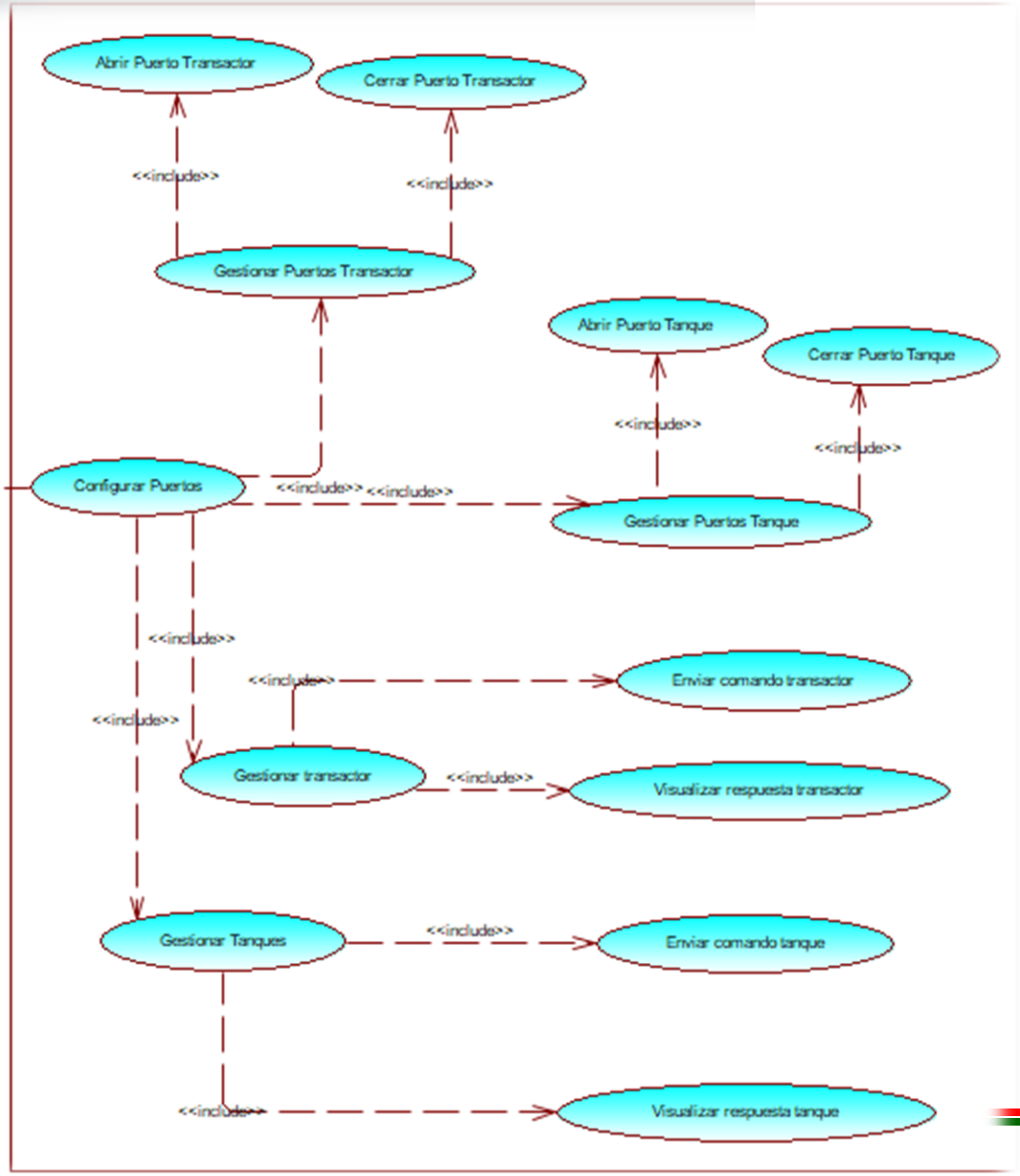
Capa 7: Personas

Desarrollo del Prototipo

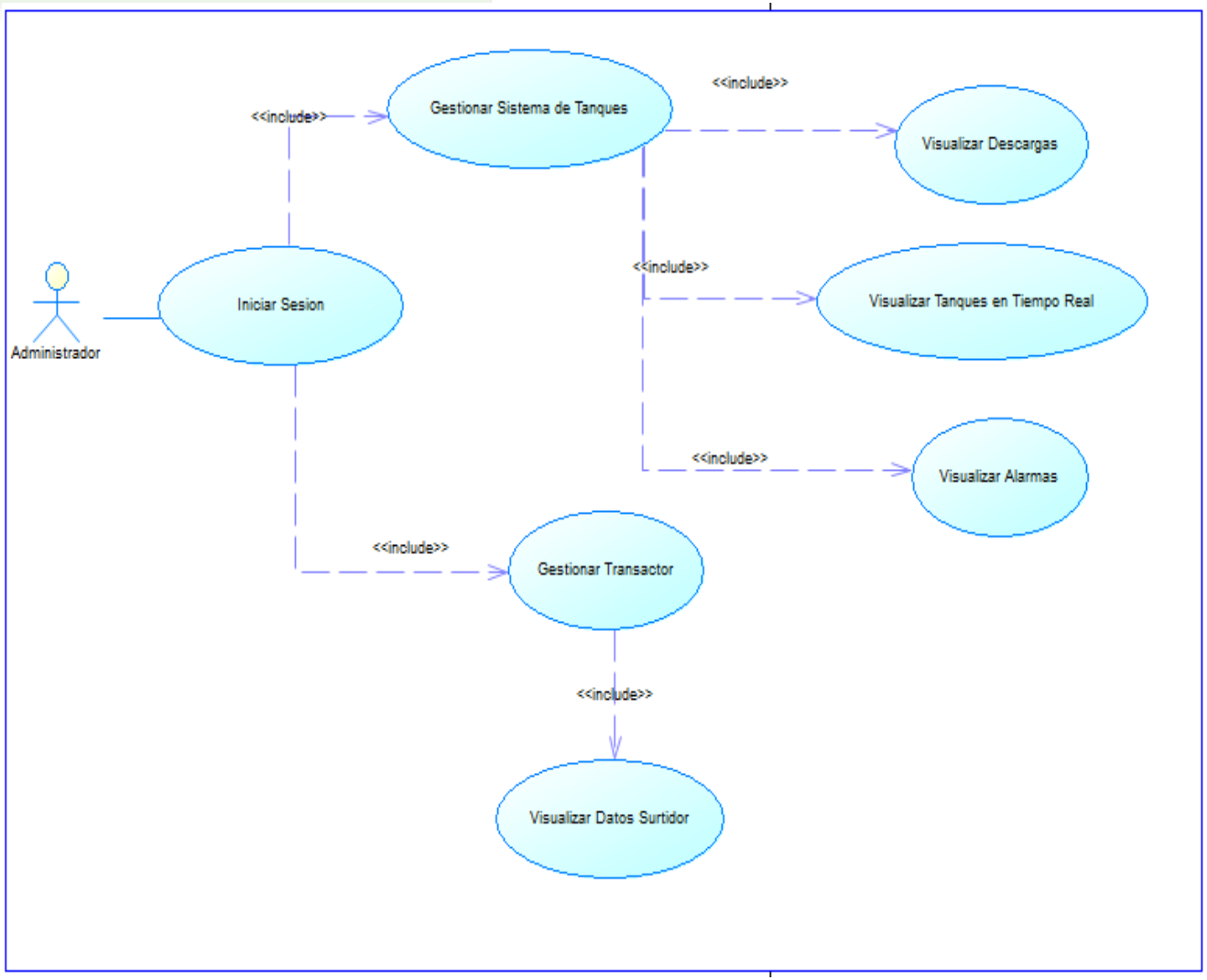
- Modelos de Casos de Uso
- Modelos de Casos de Uso Aplicación Local “AdvancedOne”
- Modelos de Casos de Uso Aplicación Web “AdvancedOne”
- Especificación Requerimientos Software
- Requerimientos Funcionales y no Funcionales



Desarrollo del Prototipo



Desarrollo del Prototipo



Desarrollo del Prototipo

Raspberry Pi 4

RS232

Veeder - Root

Transactor Pro 450

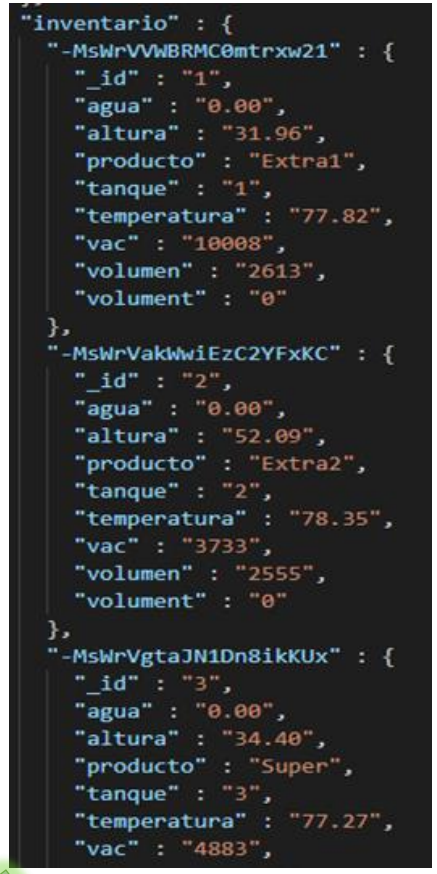
Simulador Tokheim

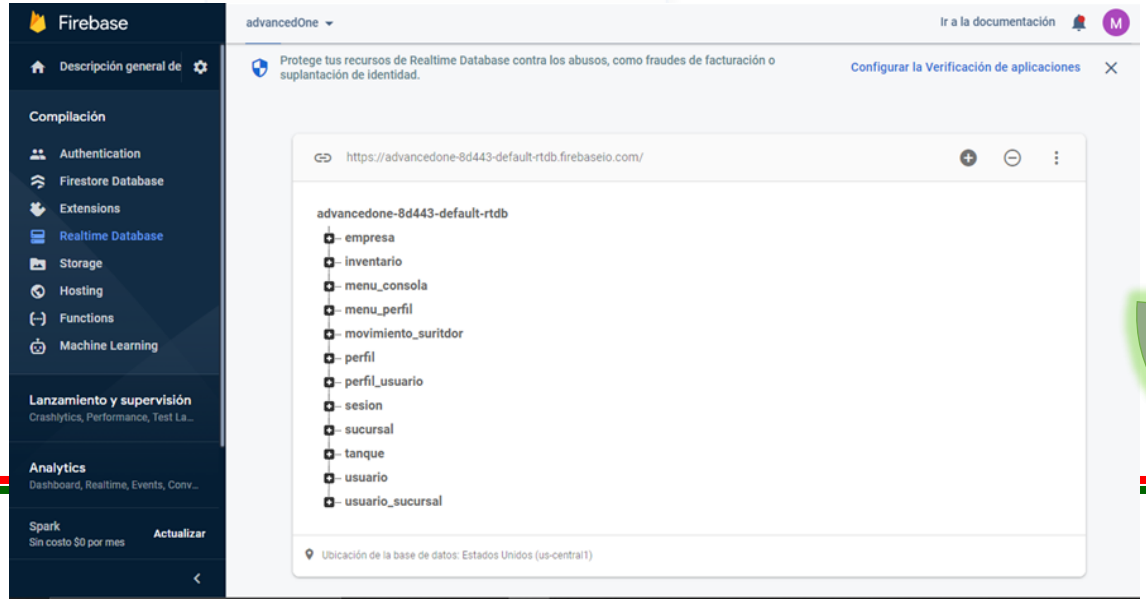
Desarrollo del Prototipo

Web Application (Angular)

Rasperry Pi/ Local App (Python)

Firebase

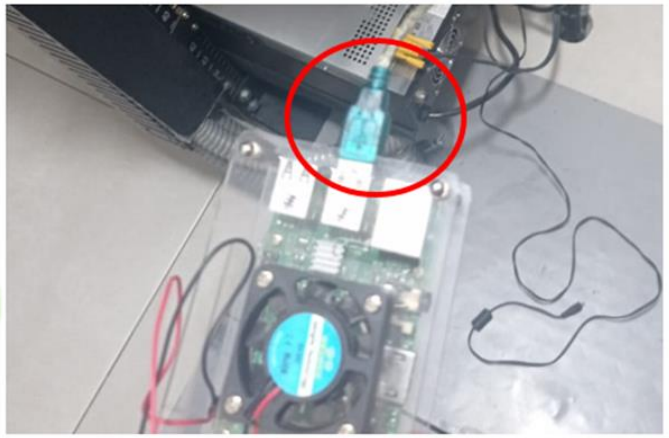




Desarrollo del Prototipo

Módulo de conexión y enlace

	PIN	
Port 2	1	(TXD) Datos transmitidos desde la consola
	2	(RXD) Datos recibidos por la consola
	5	(GND) Señal de tierra



```
# Método para definir el sistema operativo y que puerto se va a usar

def inicializar_puerto():
```

Desarrollo del Prototipo

Módulo de comunicación

```
def __init__(   self,
                data_q, output_q, error_q,
                port_num,
                port_baud,
                port_stopbits=serial.STOPBITS_ONE,
                port_parity=serial.PARITY_NONE,
                port_timeout=0.01):
```

Intercambio de datos

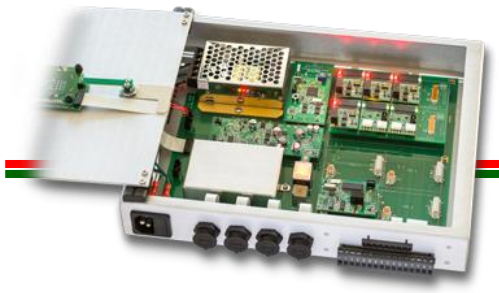
```
# INICIO OPERACIONES TRANSACTOR
# ------------------MÉTODO PARA ABRIR EL PUERTO DEL TRANSACTOR-
def on_transactor_open_button(self, com, baud):
```

```
def open(self,port=None,baud=None):
```

```
# INICIO DE OPERACIONES TANQUE
# ------------------MÉTODO PARA ABRIR EL PUERTO DEL TANQUE-
def on_tank_open_button(self, com, baud):
```

```
def initialize_RT(self):
```

```
def initialize_i201(self):
```



Desarrollo del Prototipo

Módulo de transformación y gestión de tramas

Datos en bajo nivel

```
def writeCommand(self,command=None):
```

```
command=TransactorCommandos()
```

```
self.tx_q.put(command.toString())
```

```
command=TanqueComandos()
```

Desarrollo del Prototipo

```
def update(self):
```

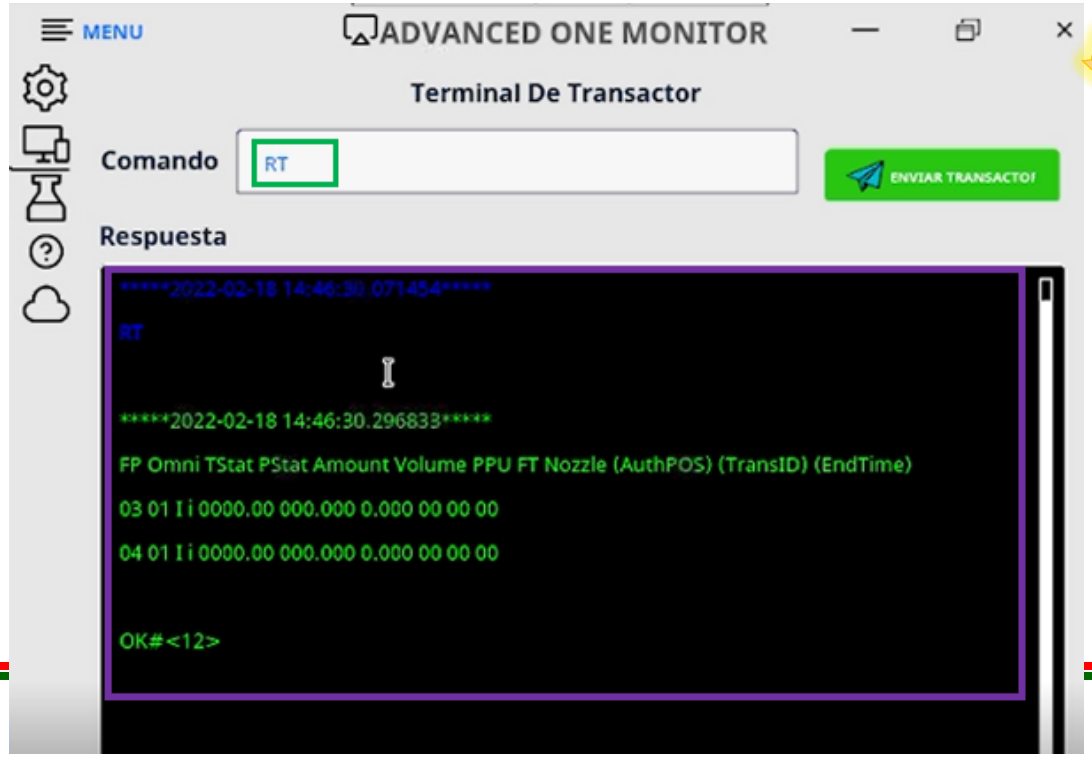
```
newdata=''.join([i[0].decode() for i in get_all_from_queue(self.rx_q)])
```


```
def onTimer(self):
```

```
def onTimer(self):
```

```
self.tank.convertirTrama(cadena)
```

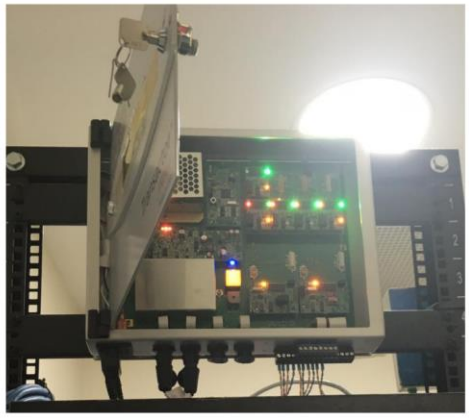
```
self.tank.recorrerCadena(cadena)
```



Desarrollo del Prototipo

Módulo de procesamiento de información en tiempo real

```
self.timer = QTimer()
```




```
self.timer.start(1)
```

```
def initialize_RT(self):
```

```
def initialize_i201(self):
```

```
def onTimer(self):
```

```
self.event_handler.ClearFinishedProcesses()
```

```
if self.transactor.alive:
    self.transactor.update()
```

```
if self.tank.alive:
    self.tank.update()
```

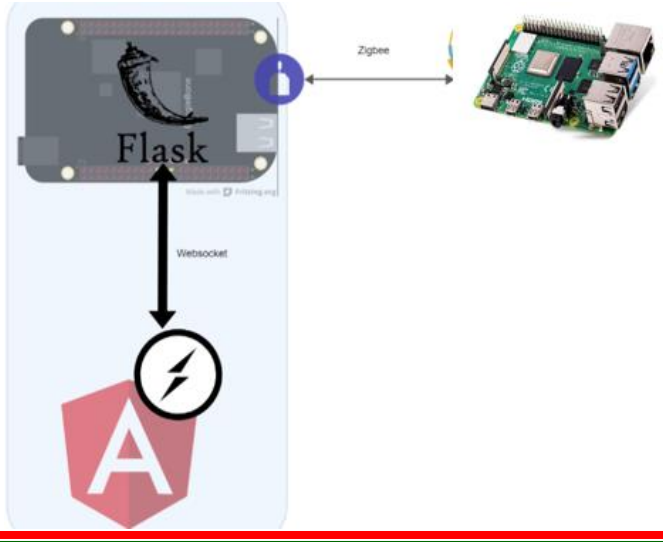
```
self.event_handler.ProcessTransactorInput(response)
```

Desarrollo del Prototipo

Módulo de procesamiento de información en tiempo real

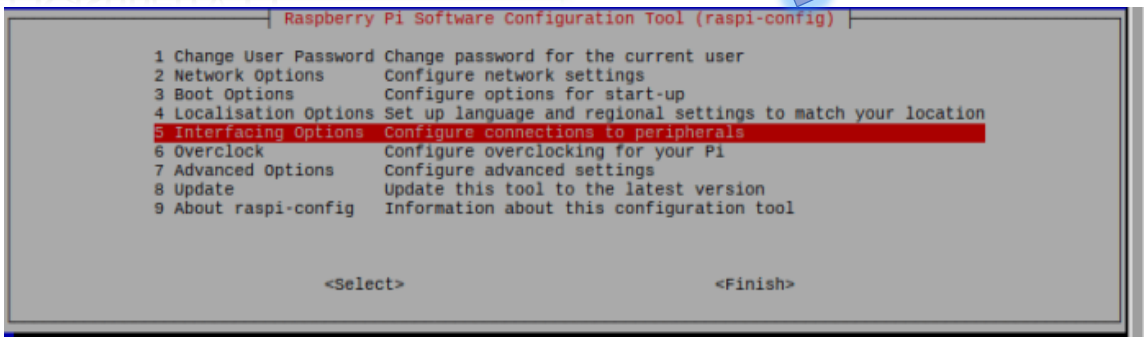

```python
from flask import Flask, render_template
from flask_socketio import SocketIO

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app)

if __name__ == '__main__':
    socketio.run(app, host='0.0.0.0', debug=True)
```

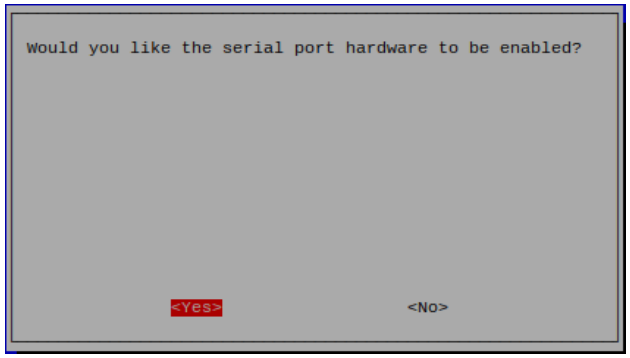
```javascript
private listenSockets() {
    this.io.on('connection', client => {

        // Listen Events

        socket.connectUser(client, this.io);

        socket.disconnected(client, this.io);

        socket.getUsers(client, this.io);

        socket.message(client, this.io);
    });
}
```

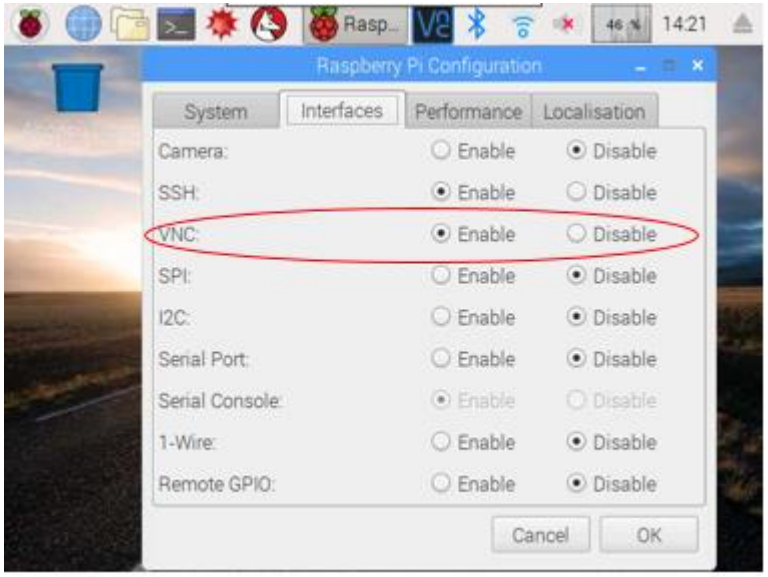


Validación del Prototipo

Configuración del Raspberry PI

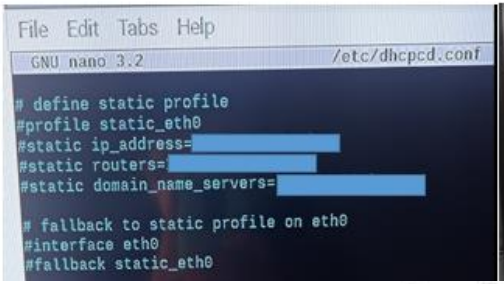


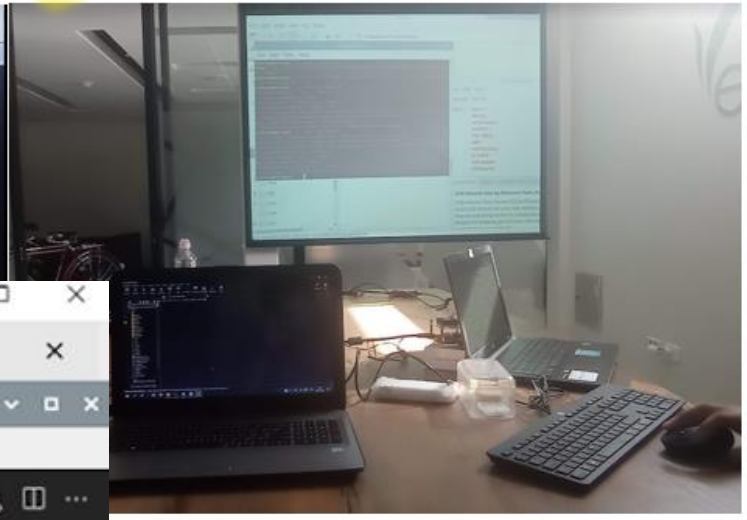


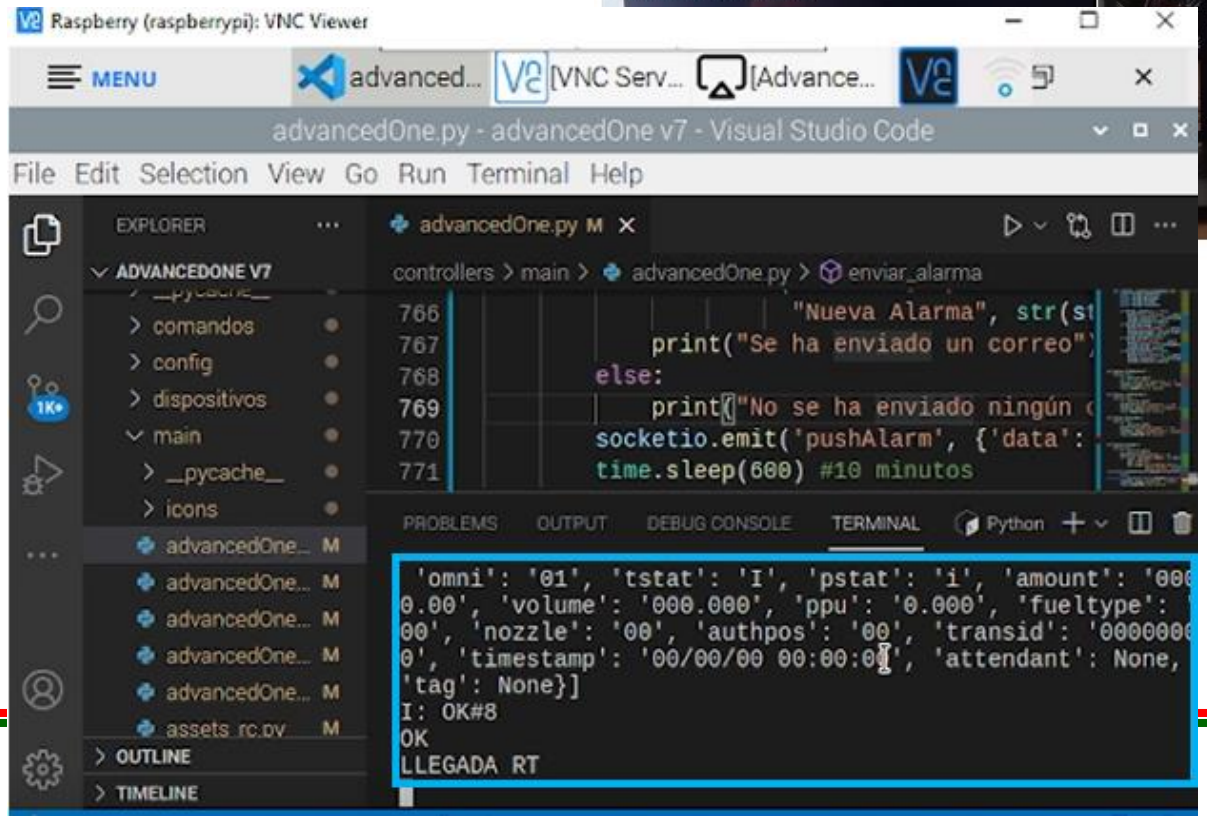


Validación del Prototipo

Configuración IP estática

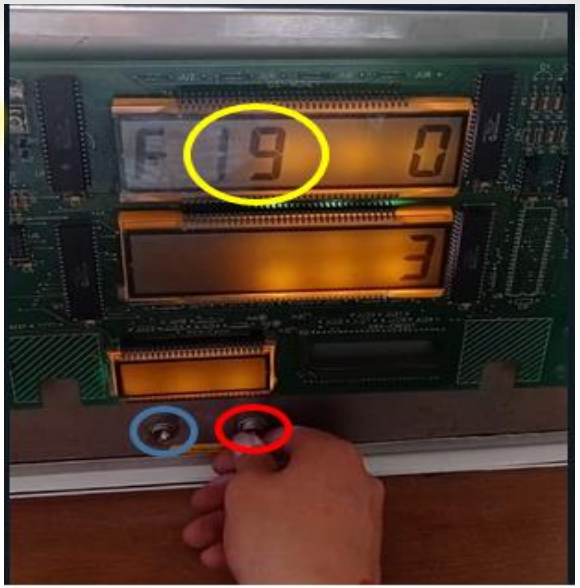






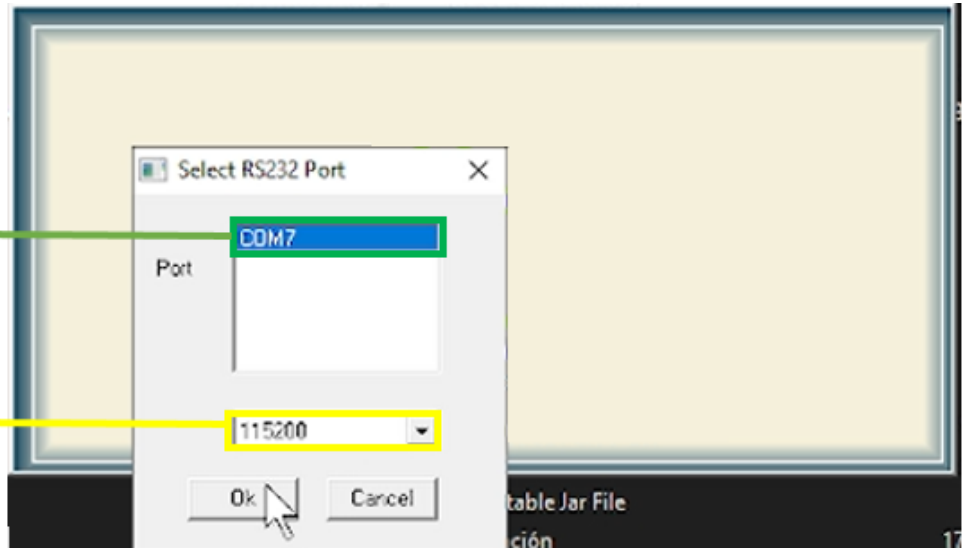
Validación del Prototipo

Tokheim



Sistema Integrador

Validación del Prototipo



Puerto

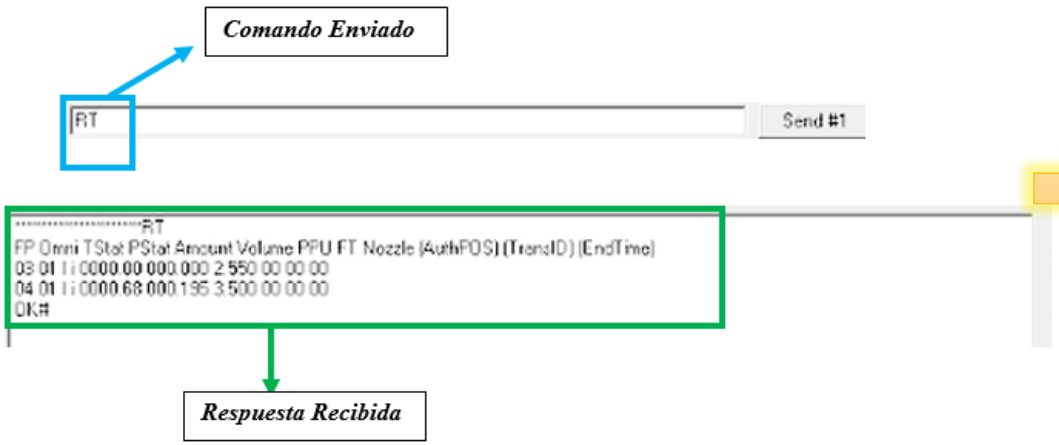
Baudios

Lados configurados

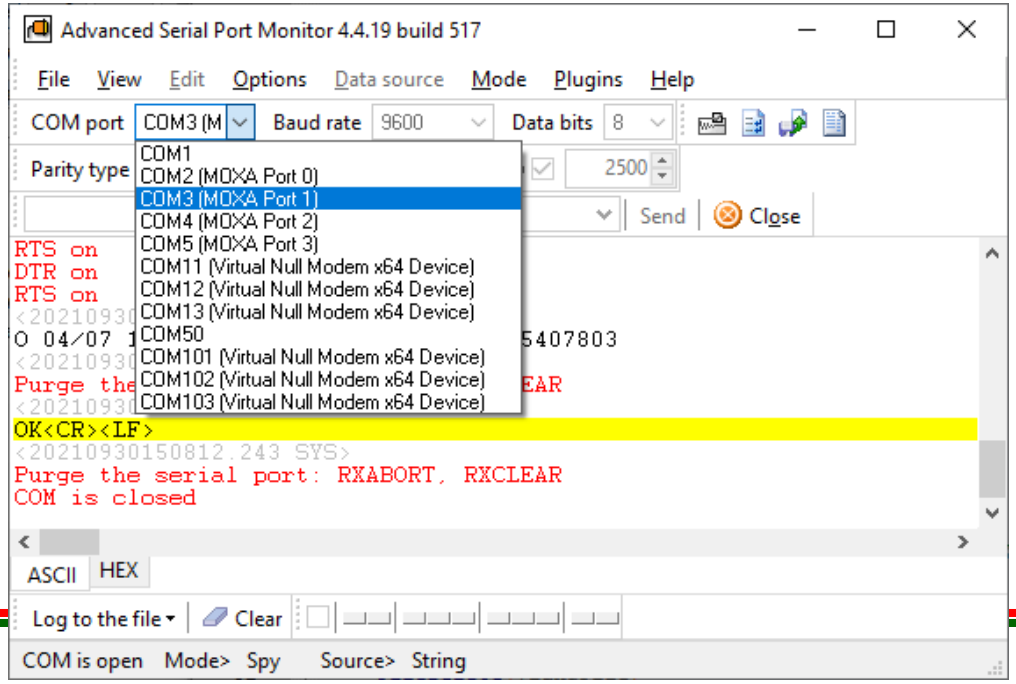
Para la configuración de los lados

Tipos de producto

Validación del Prototipo



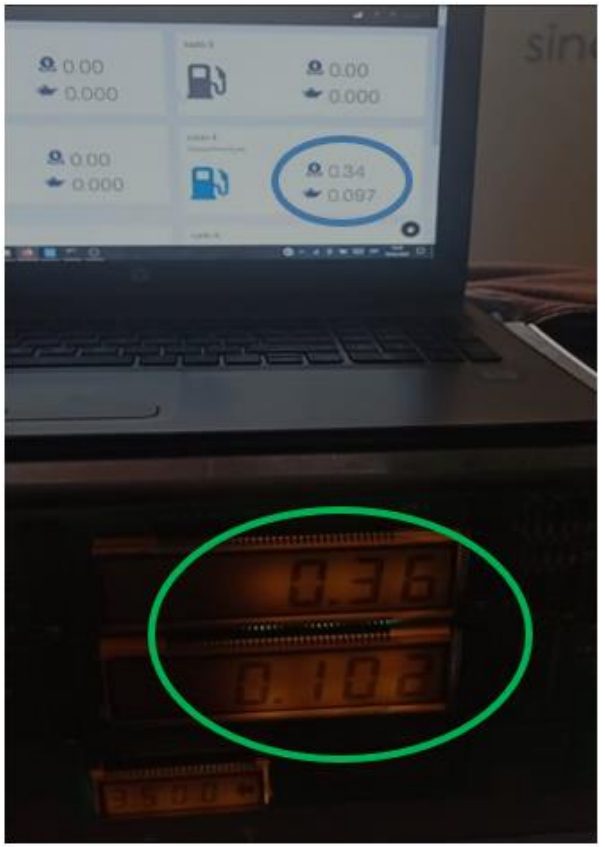
Aplicativo Transactor

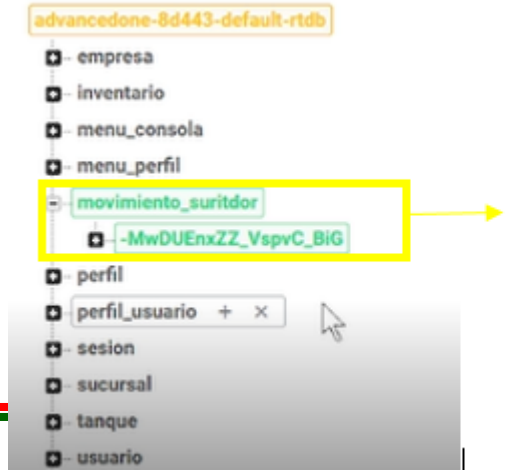


Aplicativo Sistema de Tanques

Resultados

- Integración del Raspberry con los dispositivos Transactor y Veeder-Root
- Raspberry Pi como solución de bajo costo
- Resultado del uso de la Arquitectura IOT de 7 capas
- Integración del software AdvancedOne con una base de datos No Relacional





Visualización de un nuevo dato guardado al momento

Conclusiones

•El sistema integrador de información en tiempo real que se diseñó, entre los dispositivos Transactor y Veeder-Root, con el Raspberry Pi, fue desarrollado con éxito, debido a que, se logró gestionar y controlar el combustible, además de, ofrecer una alternativa de bajo costo a la infraestructura tecnológica existente en la estación de servicio “ECO EL OASIS”, gracias a la ayuda de las tecnologías IoT y al desarrollo del software basado en la interacción de Python y Angular.

•En el desarrollo del módulo de comunicación, se pudo evidenciar que la codificación y el procesamiento de datos, tanto en el Transactor, como en el sistema de medición de tanques Veeder-Root, es bastante similar, ya que, su funcionamiento depende del envió de comandos para que dichos elementos emitan una respuesta, con la diferencia que, el Transactor responde con caracteres tradicionales, exceptuando su validador checksum, el cual, se encuentra en código ASCII, mientras que, el sistema de tanques, actúa dependiendo del formato del comando, ya que, este puede enviar la respuesta en formato hexadecimal, ocasionado un enlace con un módulo de transformación distinto al del Transactor.

•El módulo para la gestión y procesamiento de las tramas, referentes a las operaciones de control y venta de combustible, es el punto esencial del proyecto, y debe ser llevado a cabo con suma precaución, debido a que, todos los datos deben ser procesados correctamente; tanto el emisor, como el receptor son determinantes, y deben actuar de manera sincronizada, de tal forma que, cada vez que exista el envío de un comando este responda inmediatamente, con el fin de que no exista perdida de información al momento de ejecutar el poleo y de visualizar en la aplicación web.

Conclusiones

- Para el procesamiento de la información en tiempo real, la tecnología Socket IO jugó un papel fundamental, en conjunto con los Threads, ya que, ayudaron para el enlace y división de tareas, obteniendo una comunicación exitosa entre la aplicación de escritorio y la aplicación web, permitiendo que el control de combustible y el proceso de ventas, se visualice de manera inmediata por el usuario, con una de delay de 1 segundo.

- El análisis comparativo realizado entre los tres modelos de arquitectura IOT, ayudó a la construcción de un software estable, confiable y versátil para la estación de servicio “ECO EL OASIS”, ya que, dadas las características del modelo de 7 capas seleccionado permite albergar y encajar todos los componentes planteados desde el inicio del proyecto, además de poder remplazar cualquier componente en cualquier capa siempre que cumpla con las necesidades de la arquitectura.