BACKEND MQTT
By - Sir Bastidas

Base de datos

El backend necesita disponer de una base de datos para guardar la información esencial para el manejo y control de los scooters, los usuarios de la app, los pagos y todos los datos que necesiten ser almacenados. Por eso, la base de datos debe tener un sistema sólido, sostenible y sobre todo, seguro para la información sensible de los usuarios, como los datos relacionados con los pagos y sus contraseñas de sesión.

Sugiero que como base de datos se use 'PostgreSQL' por su seguridad, estabilidad, por ser uno de los estándares y por ser bastante sólido para proyectos a gran escala.

y para datos en tiempo real como la ubicación se sugieren 2 tecnologías, mongoDB y InfluxDB, yo recomiendo influxDB por su mayor desempeño en datos de tiempo real, como puede ser la ubicación, aunque mongo también es una gran alternativa en la actualidad, porque tiene un mejor manejo de la cpu, pero un manejo más deficiente en la memoria ram.

Para datos de caché, tal como se sugiere en el documento que me fue proporcionado, sería óptimo el uso de 'Redis'.

para la monitorización de los datos en tiempo real podría usarse grafana.

Seguridad de la base de datos

Uno de los aspectos fundamentales para una base de datos eficiente es el manejo adecuado de los datos de los usuarios, lo cual es esencial para el correcto funcionamiento del servidor y la base de datos.

Como práctica fundamental, se deben encriptar las credenciales de los usuarios, y los datos bancarios almacenados en la base de datos deben ser procesados de manera segura y adecuada.

Además, es crucial proteger la base de datos contra ataques informáticos, como el SQL injection, para garantizar la seguridad tanto de la empresa como de sus usuarios. Esto puede lograrse mediante un adecuado procesamiento y sanitización de los datos ingresados en la base de datos.

Servidor

Tecnología:

El backend puede crearse con Django y Django Rest Framework en el lenguaje de Python, gracias a que es un framework bastante sólido, seguro y que ofrece un flujo de trabajo cómodo para el equipo de desarrollo. Django tiene una gran variedad de herramientas para múltiples funcionalidades de control y administración del servidor, como seguridad, autenticación, gestión de usuarios, administración de datos, paneles de administrador, gestión de permisos.

Se operará por medio de un servicio principal (construido en Django / Django Rest Framework) que será el servidor base encargado del procesamiento de pagos, el monitoreo de los scooters, la gestión de usuarios, el control y administración de todo el sistema, pero que para recibir la información de los scooters estará conectado con un microservicio que haremos en Fast API que será el que se encargará de la conección con los scooters, este microservicio en un futuro será migrado a otro lenguaje y framework de programación en actualizaciones futuras, migrando a **Drogon** o **Crow** de **C++,** o su alternativa sería **Actix Rust.**

El motivo principal por el cual no se empezará directamente el primer microservicio en C++ o Rust, es debido a las limitaciones de tiempo para entregar el proyecto, ya que Rust y c++, tienen una curva de desarrollo mucho más complicada tediosa y larga, por lo que el proyecto tardaría más en salir si se empezara directamente de esta forma.

Conexión con MQTT:

El medio para conectar el servidor (Fastapi / Drogon / Crow / Actix) con MQTT sería por medio de un cliente MQTT. El más popular para Python es `paho-mqtt`, Para Rust y C++ *Rumqtt* y Eclipse Paho MQTT Este se conectará directamente con el broker MQTT que sea elegido por el encargado del IoT. Un ejemplo podría ser el broker Mosquitto o el broker EMQX. El broker será el encargado de ser el intermediario entre los scooters y el backend, así como los dispositivos móviles de los clientes tanto iOS como Android, permitiendo la administración y gestión del sistema a través del servidor.

Conexión con PostgreSQL, InfluxDB y Redis (Bases de datos):

El Backend se conectará con las bases de datos para llevar una gestión y control sobre los datos de los Scooters, Las cuentas de los usuarios y los pagos.

Se encargará de almacenar los datos principales que pueden ser gestionados en una base de datos relacional, como la información principal de los scooters, las cuentas de los usuarios, los datos de pago y las estaciones de carga. Se debe usar una usar "Psycopg2" para la conexión entre Django y PostgreSQL.

InfluxDB se encargará de los datos de estados en periodos de tiempo de cada uno de los Scooters como por ejemplo, ubicación, estado y la velocidad de los dispositivos en tiempo real. Para una conexión óptima se requiere el uso de bibliotecas de terceros y hay varias opciones como "`Django-InfluxDB y InfluxDB-client`", y aunque la conexión entre Fastapi e InfluxDB es un proceso complicado, es una de las mejores formas de manejar datos en periodos de tiempo de forma eficiente.

Redis se usará para gestionar los datos en caché, optimizando así el rendimiento y la velocidad de acceso a la información más frecuentemente consultada y su conexión con Django no tiene una gran dificultad.

Seguridad Y Autenticación

JSON WEB TOKENS (JWT): Es un mecanismo de autenticación segura entre el usuario y el servidor, que asegura que los datos no se han alterado durante el proceso. Es ideal para aplicaciones a gran escala debido a su confiabilidad y seguridad. Se puede implementar de manera efectiva con Django Rest Framework, proporcionando un alto nivel de protección para las sesiones de los usuarios.

JWT actúa como un pase entre el cliente y el servidor. Cuando un usuario inicia sesión, el servidor crea un token firmado y cifrado con información del usuario y lo envía al cliente, y cada vez que el cliente intenta autenticarse, el servidor verifica que el token sea auténtico y, si es válido, permite el acceso.

PROTECCIÓN CONTRA INYECCIONES: Las inyecciones de código son un tipo de ataque en el que un hacker introduce código malicioso en una aplicación vulnerable. Este código puede ser ejecutado por el servidor o base de datos, los tipos de inyecciones más comunes son SQL injection, Command Injection, Script injection y ataques RCE.

Si no se protege el sistema para este tipo de vulnerabilidades, los riesgos son que hackers puedan obtener acceso al sistema y la base de datos incluyendo datos bancarios, credenciales y contraseñas, por eso es de vital importancia proteger la seguridad del sistema.

Las medidas principales para proteger contra este ataque son:

- Validación de entrada de datos.
- Sanitizar entradas de datos.
- Uso de ORM.
- Protección contra XSS.
- Monitorización del servidor.
- Correcto manejo de los privilegios.

La validación y sanitización de los datos tiene como objetivo que en la entrada de los datos, el sistema verifique y procese de manera correcta la información que ingresa a la base de datos.

El uso de un ORM (Object-Relational Mapping) ayuda al correcto procesamiento de los datos y ayuda a un desarrollo más ágil.

Monitorización del servidor, monitorizar las consultas de la base de datos permite detectar actividad sospechosa y detenerla a tiempo.

El correcto manejo de los privilegios, hay que administrar correctamente los permisos y privilegios que tiene cada área del servidor y de los campos de usuario para el control de la base de datos por ejemplo, la parte del servidor encargada del inicio de sesión, no debe más que usar permisos de lectura y seria ilogico que tuviera permisos de escritura o de ejecución.

Encriptación: Encriptación: La encriptación de los datos sensibles de la base de datos, permite que en casos extremos, los datos privilegiados de la empresa y los usuarios no puedan caer en posesión de atacantes. Existen diferentes algoritmos de encriptación, simétricos (como AES) y asimétricos (como RSA), que ofrecen distintos niveles de seguridad y rendimiento. Implementar una encriptación robusta, tanto en reposo como en tránsito (usando HTTPS), es crucial para mitigar el impacto de una brecha de seguridad.

TLS:TLS es un componente fundamental para la seguridad en internet, proporcionando una base sólida para la comunicación confidencial y segura entre aplicaciones. Su implementación, especialmente a través de HTTPS, es esencial para proteger la información sensible de los usuarios.

Cloudflare: Cloudflare es la red de protección contra ataques DDoS más grande de la actualidad y uno de los CDN más rápidos, mejorando la experiencia de los usuarios.

Actúa como intermediario entre el cliente y el servidor, ofreciendo protección contra múltiples tipos de ataques, además de facilitar la implementación de HTTPS.

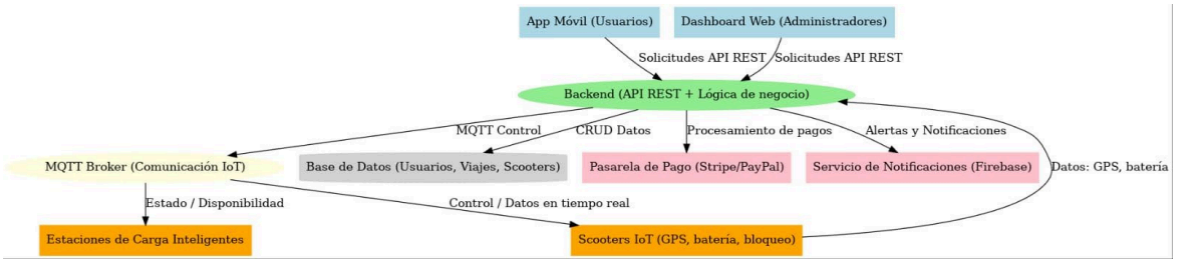
OAuth2.0: Permite a los usuarios iniciar sesión por medio de aplicaciones externas, como iniciar sesión con sus cuentas de Google, Facebook, Twitter o mediante cualquier servicio que ofrezca la posibilidad de utilizar un servicio de OAuth2.0.

Gestión del proyecto

Para manejar el proyecto de forma óptima, es crucial mantener un control efectivo entre las etapas de desarrollo, producción y las diferentes partes del proyecto. La mejor manera de lograr esto es mediante el uso de un repositorio en Git, utilizando plataformas como GitHub o GitLab.

Git es esencial porque permite separar claramente el entorno de desarrollo del entorno de producción, asegurando que los cambios sean probados y revisados antes de ser implementados en la versión final del producto. Esto minimiza errores y asegura una mayor estabilidad en producción.

Además, Git es fundamental para el trabajo en equipo. Permite a los diferentes desarrolladores colaborar de manera eficiente, trabajando en paralelo en distintas características o correcciones. Con Git, cada desarrollador puede hacer cambios en su propia rama y luego fusionar esos cambios de manera controlada, evitando conflictos y asegurando que el código final sea cohesivo y funcione correctamente.



Fases del desarrollo

1 - Creación y conexión del microservicio con los scooters mediante el protocolo de comunicación MQTT

Este será uno de los procesos más complejos de todo el desarrollo y una de las que más tiempo tomará, debido a que los scooters, cuentan con más de 100 comandos diferentes, los cuales toca programar todos 1 por 1, para que funcione tanto la gestión de los scooters, a continuación procederé a los documentos con las más de 100 funcionalidades que me toca implementar, basado en todo esto tengo que programar cada una de las funcionalidades para que funcione el sistema con los scooters. Tiene que tomarse en consideración que el término “Microservicio” puede ser un nombre engañoso, ya que no indica simplicidad si no que hace referencia a que su única función es la comunicación y registro con los scooters mediante los comandos MQTT. ya que el resto del sistema estará hecho en DJango y recibirá los datos a partir de que los envíe este microservicio.

Power on (WR)

Topic: Vehicle number
QoS : 0
Payload:
{
"a": 1 // integer , unlock
}
Example: If you want to turn on the scooter, send {"a":1}

Power on ack (RD)

Topic : bike
QoS : 0
Payload:
{
"a": 2, // integer , Confirm power on
"i": "12AB", // string , Vehicle number
"t": 1503046415, // long,random number
"s": <status>, // integer ,status： 0 :success; 1 :hardware fault; 2 : Firmware upgrading
"n": 0 // integer , power on serial number
}

Power off (WR)

Topic: Vehicle number
QoS : 1 or 2
Payload:
{
"a": 3// integer , Shutdown
}
Example: if you want to turn off the scooter, send {"a":3}

Power off ack(RD)

Topic : bike
QoS : 0
Payload:
{
"a": 4, // integer , Confirm shutdown
"i": "12AB", // string , Vehicle number
"t": 1503046415, // long,random number
"s": <status>, // integer ,Status： 0 :success; 1 :hardware fault; 2 : Firmware upgrading
"n": 0 // integer, Shutdown serial number
}

Clear single riding mileage (WR)

• Instruction format：

Topic: Vehicle number

QoS : 0

Payload:

{

"a": 5 // integer

}

Example: If you want to clear single riding mileage, send {"a":5}

Clear single riding mileage ack(RD)

Topic: bike

QoS : 0

Payload:

{

"a": 6,// integer

"i": "12AB", // string , Vehicle number

"t": 1503046415, // long ,random number

"s": <status>, // integer ,Status： 0 :success; 1 :hardware fault; 2 :Firmware upgrading ;3: It is already later than the latest shutdown time

"n": 0 // integer serial number

}

Clear single riding time (WR)

•Instruction format:

Topic: Vehicle number

QoS : 0

Payload:

{

"a": 7 // integer

}

Example: if you want to clear single riding time, send {"a":7}

Clear single riding time ack (RD)

Topic: bike

QoS : 0

Payload:

{

"a": 8, // integer

"i": "12AB", // string , vehicle number

"t": 1503046415, // long,random number

"s": <status>, // integer , Status：0 :success; 1 :hardware fault; 2 :Firmware upgrading ;3: It is already later than the latest shutdown time

"n": 0 // integer serial number

}

Clear total mileage (WR)

•Instruction format:

Topic: vehicle number

QoS: 0

Payload:

{

"a": 9 // integer

}

Example: if you want to clear total mileage, send {"a":9}

Clear total mileage ack (RD)

Topic: bike

QoS : 0

Payload:

{

"a": 10,// integer

"i": "12AB", // string , vehicle number

"t": 1503046415, // long,random number

"s": <status>, // integer ,Status：0 :success; 1 :hardware fault; 2 :Firmware upgrading ;3: It is already later than the latest shutdown time

"n": 0 // integer serial number

}

Clear the total riding time (WR)

• Instruction format:

Topic: vehicle number

QoS : 0

Payload:

{

"a": 11 // integer

}

Example: if you want to clear the total riding time, send {"a":11}

Clear the total riding time ack(RD)

Topic: bike

QoS : 0

Payload:

{

"a": 12,// integer

"i": "12AB", // string , vehicle number
"t": 1503046415, // long,random number
"s": <status>, // integer , Status：0 :success; 1 :hardware fault; 2 :Firmware upgrading ;3: It is already later than the latest shutdown time
"n": 0 // integer serial number
}

Set the speed limit data (WR)

•Instruction format:
Topic: vehicle number
QoS : 1
Payload:
{
"a": 13, // integer
"k": 2025 // integer 2000---2030
}
Example: if you want to set the speed to 20km/h, send {"a":13,"k":2020}
The numerical value after K is 2000+ X km/h

Set the speed limit data ack (RD)

Topic: bike
QoS : 0
Payload:
{
"a": 14, // integer
"i": "12AB",// string , vehicle number
"t": 1503046415, // long,random number
"s": <status>, // integer , Status： 0 :success; 1 : fault
}

Query vehicle parameters (WR)

•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{
"a": 15 // integer
}
Example: if you want to get the vehicle parameters, send {"a":15}

Query Vehicle parameter ack (RD)

Topic: bike
QoS : 0
Payload:
{
"a": 16 , // integer
"i": "12AB",// string , vehicle number
"s ": 8 , // integer Status
"k ": 8 , // integer speed limit values KM
"p ": 8 , // integer current speed KM
"e ": 8 , // integer Vehicle error code (0 :no fault; 1:motor hall fault; 2: speed throttle fault; 4 :motor stalling; 5:over-current; 6: motor over temperature; 7: controller over temperature; 8: battery undervoltage; 9:battery overvoltage; 10: communication failure)
"b ": 8 , // battery (battery percentage)
"y": 8 , // integer total riding time seconds
"q": 8 , // integer total riding mileage KM
"w": 8 , // integer single riding time seconds
"z": 8, // integer single riding mileage KM
"c": <status> // integer , status: 0 : power on ;1 : shutdown
}

Query GPS location data (WR)

Topic: vehicle number
QoS : 0
Payload:
{
"a": 18 // integer, issue the gps command
}
mark: if you want to get the longitude and latitude, send {"a":18}. But this command is not usually used. Because when the scooter is positioning successfully, it will actively report the position alternately. The default GPS reporting interval is about 30 seconds per time. It can be set through the command 33.

Query GPS location data ack a (RD)

Topic: bike
QoS : 1
Payload:
{
"a": 19, // integer
"i": "12AB", //string, vehicle number
"g": "latitude, longitude, business_type, location_type",
//"3020.5887,12006.4229,1" business_type:0 Normal ,1 alarm , location_type : 0 GPS,1 LBS
"t": 1503046415 // long GPS reporting time
}

Mark: This command will send to mqtt automatically when the iot gets valid gps data.

Reboot IOT (WR)

•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{
"a": 20 // integer
}
Example: if you want to reboot the IOT, send {"a":20}

Qurey hardware and firmware version (WR)

•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{
"a": 21 // integer
}
Example: if you want to get the hardware info of network mudule, send {"a":21}

Qurey hardware and firmware version ack (RD)

Topic: bike
QoS : 1
Payload:
{
"a": 22, // integer
"i": "12AB", //string, vehicle number
"h": hardware version
"f": software version
"c":simcard number
"m": IEMI number
"s": vehicle status: 0 :Powered on, 1 : Powered off
"t": 1503046415 // long, random number
}

Start Upgrade firmware (WR)

The server inform scooter to upgrade firmware

•Instruction format：

Topic: vehicle number

QoS : 0

Payload:

{

"a": 23 ,// integer,

"l": 50 ,// integer, package number of updating file, 1024 bytes each package

}

Start Upgrade firmware (RD)

Topic : bike

QoS : 1

Payload:

{

"a": 66, // integer ,

"i": "12AB",// string , vehicle number

"s": 0, // integer， 0 can be upgraded 1 prohibit to upgrade

}

Send updating data(WR)

•Instruction format：

Topic: vehicle number

QoS : 0

Payload:

{

"a": 67, // integer,

"x":55, // integer,firmware package serial number

"l":55, // integer,data length

"d":012z//firmware data binary type

}

Send updating data ack(RD)

Topic : bike

QoS : 1

Payload:

{

"a": 68, // integer,

"i": "12AB", // string ,vehicle number

"x":55, // integer,firmware package serial number

"s": 0, // integer, 0 receive the firmware package successfully(can issue send next data packet) 1 failed to receive the firmware package（at this time the server need to resend this packet of data.

When failed more than 5 times, the module will reboot for re-upgrading)
}

Updating success report(RD)

Topic : bike
QoS : 1
Payload:
{
"a": 69, // integer ,
"i": "12AB", // string , vehicle number
"s": 0, // integer,0 upgrade successfully 1 upgrade failed
}

Query IEMI number(WR)

•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{
"a": 24 // integer
}
Example: If you want to get IEMI number, send {"a":24}

Qurey IEMI number ack(RD)

• Instruction format:
QoS : 0
Payload:
{
"a": 25, // integer
"i": "12AB",//string, vehicle number
"h":123456789888888,// IMEI number
"t": 1503046415 // long random number
}

Vehicle status report (RD)

•Instruction format:
Topic : bike
QoS : 1
Payload:
{
"a": 27, // heartbeat
"i": "12AB", // string , vehicle number
"b": <battery ratio >, // integer, percentage of charge, from 0-100
"c": <status> // integer , status: 0 : power on; 1 : power off
"s":1<vehicle posture> // integer , status: 0 : normal; 1 : fall down . Only valid power on status.
"r":37.5<voltage of battery> // integer
"v":25 <sensitivity of vibration> // integer
"x":0> // 1= station attached; 3= no satation integer
}
Mark: This command has been sent to server automatically.

Alarm buzzer(WR)
Topic: vehicle number
QoS : 0
Payload:
{
"a": 28 // integer , alarm
}
Example: If you want the buzzer of iot to sound, send {"a":28}
Mark: No ack for this command

Set server parameters (WR) This command has been deleted and replaced by command 101
•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{
"a": 33, // integer
"u":18.180.156.177,1844,00012,30,10,3
}

// parameters as follows
IP+PORT+vehicle number+GPS reporting interval+ heartbeat packet interval+times(numbers) of resent after network heartbeat packet failure

Example: if you want to set new IP as 123.456.789.123 , port as 1234, vehicle number as 000003, GPS reporting interval as 45 seconds, send {"a":33,"u":123.456.789.123,1234,000003,45}. When the command is executed succefully, the module will connect to the new IP, PORT. This process will take a while because IOT needs to reconnect to the network.

Set server parameters ack (RD)

Topic : bike
QoS : 0
Payload:
{
"a": 34, // integer,
"i": "12AB",// string , vehicle number
"s":0, // integer 0: success ; 1: failure
}

Query server parameters (WR)

•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{
"a": 35, // integer
}
Example: if you want to get user parameters, send {"a":35}

Query server parameters ack(WR)

Topic : bike
QoS : 0
Payload:
{
"a": 36, // integer ,
"i": "12AB", // string , vehicle number
"u":"018.180.156.177,1883,username ,password, vehicle number,30"
}
IP,IP_PORT,username,password,vehiclenumer,gps interval-gps

Lamp switch setting(WR)

Topic: vehicle number
QoS : 0
Payload:
{
"a": 37, // integer , headlight control
"d":0// integer 0: turn off the light 1: turn on the light
}

Example: This command only works when Lamp mode setting in" command cntrol mode"
after send commad {"a":43,"j":0}

Lamp switch setting ack(RD)

Topic : bike
QoS : 0
Payload:
{
"a": 38, // integer ,
"i": "12AB",// string , vehicle number
"s":0, // integer 0: Success 1:failure
}

Vibration sensitivity setting (WR)

Topic: vehicle number
QoS : 0
Payload:
{
"a": 39, // integer , Vibration setting
"v":0// integer 0: turn off vibration >0: vibration sensitivity
}

Example: vibration here is vibration of scooter caused by outside detected when the scooter is off. When the number of successive vibrations reach a value, the bell sounds and alarms, at the meantime, the scooter motor will be locked. It means motor stalling to prevent somebody to slide the scooter. If the scooter do not detect vibration in 120 seconds, it will exit this status.
The value 0 behind this command is turn off vibration. If you send the command {"a":39,"v":0}, the scooter's buzzer will not make any sound when the user moves or vibrates the scooter.
When the value behind v is >0, it is used to adjust the vibration sensitivity. Bigger value, less sensitive. The biggest value is 99. The default value is 25. If you want to set the sensitivity as 10, send {"a":39,"v":10}

Vibration sensitivity setting ack(RD)

Topic : bike
QoS : 0
Payload:
{
"a": 40, // integer ,
"i": "12AB",// string ,vehicle number
"s":0, // integer 0:success 1: failure
}

Kilometer or mile switch setting (WR)

Topic: vehicle number
QoS : 0

Payload:
{
"a": 41, // integer , Kilometer mile switch
"f":0// integer 0: kilometer 1: mile
}
Example: if you want to set speed unit as kilometer on display, send {"a":41,"f":0} when the scooter is off.
if you want to set speed unit as mile on display, send {"a":41,"f":1} when the scooter is off.

Kilometer or mile switch setting ack(RD)
Topic : bike
QoS : 0
Payload:
{
"a": 42, // integer ,
"i": "12AB",// string , vehicle number
"s":0, // integer 0: success 1: failure
}

Lamp mode setting (WR)
Topic: vehicle number
QoS : 0
Payload:
{
"a": 43, // integer , headlight switchover
"j":0// integer 0: command control 1: always on
}
Example:This command works only when the scooter is powered on. You need to send command {"a":1} turn on the scooter first.

Lamp mode setting ack ack(RD)
Topic : bike
QoS : 0
Payload:
{
"a": 44, // integer ,
"i": "12AB",// string , vehicle number
"s":0, // integer 0: success 1:failure
}

Set_Assist_Level (WR)
•Instruction format:

Topic: vehicle number
QoS : 1
Payload:
{
"a": 49, // integer
"k": 0 // integer[0~2]
}

Set_Assist_Level ack (RD)
Topic: bike
QoS : 0
Payload:
{
"a": 50, // integer
"i": "12AB",// string , vehicle number
"s": <status>, // integer , Status： 0 :success; 1 fault
}

APN setting(WR)
•Instruction Format:
Topic: vehicle number
QoS : 0
Payload:
{
"a": 53, // integer
"z":AT+QICSGP=15,1,"apn","username","password",0 // string
}
Example: If you want to set the APN as 123, username as 456, password as 789, send {"a":53, "z":AT+QICSGP=15,1,"123","456","789",0}
If there is no username and password, the relevant position is empty. Such as APN is 123, username is empty, password is empty, send {"a":53, "z":AT+QICSGP=15,1,"123","","",0}

APN setting ack(RD)
Topic : o
QoS : 0
Payload:
{
"a": 54, // integer,
"i": "12AB",// string ,vehicle number
"s":0, // integer 0: success 1: failure
}

Scooter status parameter Report(RD)
Topic : o

QoS : 0
Payload:
{
"a": 55, // integer,
"i": "12AB",// string ,vehicle number
"s":0, // integer 0: the scooter is not fall down 1: the scooter is fell down
"r" //interger means the battery voltage, unit v
}

Mark: This command is deleted in latest protocol version.

Vibration dection Report(RD)
Topic : o
QoS : 0
Payload:
{
"a": 57, // integer,
"i": "12AB",// string ,vehicle number
"s":1, // integer
}

Mark: This command is deleted in latest protocol version.

Special alarm buzzer(WR)
Topic : vehicle number
QoS : 0
Payload:
{
"a": 58, // integer,ring
"v":5,// Numbers of rings
“i”:100,// time of buzzer ring unit:ms
“L”:200 // the time of that the buzzer does not sound unit:ms
}
Annotation: The above parameter means that the buzzer will ring for 100ms, then not for 200ms, so it alternates 5 times.

Battery unlock(WR)
Topic: vehicle number
QoS : 0
Payload:
{
"a": 60 // integer
}
Example: if you want to remove battery, send {"a":60}

Battery unlock ack(RD)

Topic : bike
QoS : 0
Payload:
{
"a": 61, // integer,confirmed power on
"i": "12AB",// string ,vehicle number
"s": <status> // integer,status: 0 :success ; 1: hardware failure; 2:firmware upgrading
}

Battery lock(WR)

Topic: vehicle number
QoS: 0
Payload:
{
"a": 62 // integer
}
Example: if you want to lock the battery, send {"a":62}

Battery lock ack(RD)

Topic : bike
QoS : 0
Payload:
{
"a": 63, // integer ,confirmed power on
"i": "12AB", // string ,vehicle number
"s": <status> // integer , status: 0 :success ; 1: hardware failure; 2:firmware upgrading
}

Chainlock unlock(WR)

Topic: vehicle number
QoS: 0
Payload:
{
"a": 71 // integer
}
Example: if you want to unlock the chain lock, send {"a":71}

Chainlock unlock ack(RD)

Topic : bike
QoS : 0
Payload:

{
"a": 72, // integer
"i": "12AB", // string ,vehicle number
"s": <status> // integer , status: 0 :success ; 1: hardware failure; 2:firmware upgrading
}

Enter pause mode (WR)

Topic: vehicle number
QoS: 0
Payload:
{
"a": 73 // integer
}
Example: if you want to enter pause mode{"a":73}

Enter pause mode ack(RD)

Topic : bike
QoS : 0
Payload:
{
"a": 74, // integer ,
"i": "12AB", // string ,vehicle number
"s": <status> // integer , status: 0 :success ; 1: failed
}

Exit pause mode (WR)

Topic: vehicle number
QoS: 0
Payload:
{
"a": 75 // integer
}
Example: if you want to exit pause mode{"a":73}

Exit pause mode ack(RD)

Topic : bike
QoS : 0
Payload:
{
"a": 76, // integer ,
"i": "12AB", // string ,vehicle number
"s": <status> // integer , status: 0 :success ; 1: failed
}

Query IOT Log(WR)

Topic: vehicle number
QoS: 0
Payload:
{
"a": 77 // integer
}
Example: if you want to query IOT Log send {"a":77}

Query IOT Log ack(RD)
Topic: vehicle number
QoS: 0
Payload:
{
"a": 78 // integer
"i": "12AB", // string ,vehicle number
"b": "12AB", // string ,vehicle number
"c": "12AB", // string ,vehicle number
"d": "12AB", // string ,vehicle number
"e": "12AB", // string ,vehicle number
"f": "12AB", // string ,vehicle number
"g": "12AB", // string ,vehicle number
}

Query status of chain lock **(WR)**
Topic: vehicle number
QoS: 0
Payload:
{
"a": 83 // integer
}
Example: if you want to query status of chain lock send {"a":83}

Query status of chain lock **(RD)**
Topic : bike
QoS : 0
Payload:
{
"a": 84, // integer ,
"i": "12AB", // string ,vehicle number
"s": <status> // integer , status: 0 :close ; 1: open
}

Viberating check **(RD)**
Topic : bike

QoS : 0
Payload:
{
"a": 88, // integer ,
"i": "12AB", // string ,vehicle number
"s": <status> // integer , status: reserved}

Reboot IOT if Updated (WR)
Topic: vehicle number
QoS: 0
Payload:
{
"a": 93 // integer
}
Example: if you want to reboot the iot after you update it successfully send {"a":93}

Remark:No respone for this command.

Query 4G signal intensity(WR)
Topic: vehicle number
QoS: 0
Payload:
{
"a": 97 // integer
}
Example: if you want to query signal intensity, send {"a":71}

Query 4G signal intensity ack(RD)
Topic : bike
QoS : 0
Payload:
{
"a": 98, // integer ,confirmed power on
"i": "12AB", // string ,vehicle number
"x": // integer , level of signal intensity
}

Set server parameters (WR)

•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{
"a": 101, "u":18.180.156.177: 1883:username :password:
}

Set Iot Parameters ack(RD)

QoS : 0
Payload:
{
"a": 102, // integer ,confirmed power on
"i": "12AB", // string ,vehicle number
"s": // 0=success 1=failed
}

Modify IOT SN **(WR)**

•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{"a": 103,"u":SN_Number:}

Modify IOT SN ack(RD)

QoS : 0
Payload:
{
"a": 104, // integer ,confirmed power on
"i": "12AB", // string ,vehicle number
"s": // 0=success 1=failed
}

---

unlock the motor lock **(WR)**

•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{"a": 105}

unlock the motor lock ack(RD)

QoS : 0
Payload:
{
"a": 106, // integer ,confirmed power on
"i": "12AB", // string ,vehicle number
"s": // 0=success 1=failed
}

---

lock the motor lock **(WR)**
•Instruction format:
Topic: vehicle number
QoS : 0
Payload:
{"a": 107}

lock the motor lock ack(RD)
QoS : 0
Payload:
{
"a": 108, // integer ,confirmed power on
"i": "12AB", // string ,vehicle number
"s": // 0=success 1=failed
}

2 - Creación y conexión de la base de datos de series temporales “TSD” InfluxDB con el microservicio.

InfluxDB:

Los datos recibidos por el servidor de los scooters (datos como estado, ubicación, velocidad, nivel de batería entre otros), tienen que ser monitorizados, y controlados por los administradores del proyecto para tener un control sobre los dispositivos IoT.

Por eso es esencial que el microservicio, almacene los datos para que puedan ser controlados desde un panel de administrador.

Tiene que programarse la base de datos para que reciba, procese y retorne los datos de los scooters en la mayor velocidad posible al sistema principal en Django.

Es por eso que tiene que tomarse en consideración todos los datos que van a registrarse en la base de datos, por motivos de optimización no es conveniente que todos los datos se guarden de esta forma, pero sí los más esenciales que bajo mi opinión profesional serian

- GPS (cada 5 seg)
- Bateria (cada 20 seg)
- Velocidad (cada 5 seg)
- Estado (cada 30 seg)
- Señal (cada 15 seg)

aunque se podrían añadir registros en tiempo real extras si así lo desean los directivos (tomando en consideración las ventajas y desventajas que esto traería)

se manejan con tags que indiquen la placa de cada scooters, para así poder diferenciar entre la diferente variedad de dispositivos, pero bien el trabajo no termina solo en la creación de la base de datos si no que continua en el procesamiento de los datos para que puedan ser mostrados de una forma legible en las páginas de administración y control.

Estos datos también son vital y esenciales para el cálculo de las tarifas de los scooters, y también servirán

por ejemplo el GPS junto a la Velocidad tendrían que implementar un algoritmo de unión de nodos que calculando la velocidad, dirección, latitud, y longitud puedan ser

expresadas sus rutas de la mejor forma posible en un mapa en tiempo real (para el rastreo de los dispositivos), y también para la implementación de las geoespaciales de los scooters (para que estén limitados a zonas determinadas), se tendría que recurrir a algoritmos que implementen geometría no euclidiana, para que pueda calcularse de forma correcta las distancias y limitaciones considerando la leve pero igualmente relevante para los cálculos curvatura de la tierra.

3 - Creación y definición de las apps del sistema principal en Django Y Django Rest Framework y su base de datos relacional en “PostgreSQL”.

Django funciona por medio de pequeñas aplicaciones para dividir el trabajo y mantener de forma más organizada y limpia el proyecto.

App De Usuarios:

lo primero sería programar la aplicación de usuarios que sería la encargada de procesar todo lo que tenga que ver con usuarios, esto incluye la creación de cuentas e inicio de sesión, crear sus respectivos Breakpoints para que la app móvil pueda integrarse con esta parte de la aplicación, también se conectara con el inicio de sesión OAuth2.0, también debe tomarse a consideración y programar los Modelos de la base de datos para los usuarios, con sus respectivos campos:

USUARIOS

- Nombre
- Apellido
- Correo Electrónico
- Contraseña
- Número de teléfono (opcional)
- Fecha De Creación (automatizado)
- Saldo (Se conectará con la app de pagos)

App De Pagos:

Aquí se programaran todos los datos relacionados con los pagos y la gestión del saldo, es decir aquí se procesarán todos los pagos, datos bancarios y la recarga del saldo, esta parte del software es una de las que más trabajo lleva porque es esencial la implementación de todas las medidas de seguridad posibles y la protección de los datos. También tendrá conexión con la aplicación móvil

Base de datos: Pagos realizados

- Usuario que realizó el pago
- Método de pago
- Monto del pago
- Fecha del pago

Base de datos: Viajes

- Usuario que realizó el viaje
- placa del scooter que uso
- Distancia Recorrida por el scooter
- Ubicación Donde se movilizó el scooter (para calcular tarifas relativas)
- Monto del viaje

App De Tarjetas:

POR LA FALTA DE LAS TARJETAS EN FÍSICO, NO SE PUEDE PLANIFICAR ESTO TODAVÍA.

App De Scooters:

Esta es la app que conectará el microservicio mencionado anteriormente con la aplicación principal, y será encargado la conexión entre los 2 softwares, y el procesamiento para enviar y recibir datos puntuales de los scooters (también para enviarlos al menú de administrador y gestionar los scooters desde el panel), en resumen estará encargado de recibir la información que ya tratamos en la fase 1,

También esta será la app encargada del escaneo de los QR, y el alquiler de estos mismos.

App De Administrador(es):

Esta es la app que se encargará del monitoreo control, gestión y manejo de los scooters, los usuarios, la información y todas las funcionalidades del software y los scooters, desde aqui se podran ver la ubicación, bloquear, controlar, administrar, gestionar, subir o bajar tarifas, manejar absolutamente todo.

También tendrá la opción de exportar los datos a excel si así se desea.

También será el nexo entre el resto de aplicaciones y será el núcleo de todo el sistema.

por motivos de seguridad prefiero aislar los usuarios de administrador y los de clientes (este modelo no tendrá Oauth2.0 porque no es necesario)

Base de datos: Usuarios Admin

- Nombre
- Apellido
- Correo electrónico
- Contraseña
- Permisos del usuario
-

Base de datos: Registro de cambios de usuarios admin

- Usuario Admin
- Cambio Realizado (ej. Jose Hernandez suspendio una cuenta)
- Correo electrónico
- fecha y hora del cambio realizado

4 - Fase de pruebas, optimización y solución de errores en el código y preproducción.

En esta fase se enfoca en todas las pruebas de calidad y control que va a tener el servidor y las bases de datos para que estén pulidas y se consideren terminadas.

También se escoge entre las múltiples opciones un alojamiento para el servidor, como Amazon AWS, Microsoft Azure, Google Cloud o algún otro servicio VPS, y se empezará con las configuraciones e implementaciones necesarias para que el servidor funcione, también es importante remarcar que serán utilizados contenedores Docker para agilizar el proceso y muy probablemente kubernetes.

también se agregaran las implementaciones de seguridad de DNS y Dominio, se usará cloudflare, y se configurarán todos los parámetros finales para que el sistema funcione

5 - Producción (Proyecto terminado)

Se subiria todo el servidor al alojamiento establecido y estaría terminado, solo se realizarán unas últimas comprobaciones y

Ya estaría terminado el servidor, las bases de datos y todo lo necesario para que los scooters entren en funcionamiento.

por último solo quedaría las futuras actualizaciones, el soporte y el mantenimiento.