TAIST ICT 720

# DEPARTMENT STORE SMART PARKING SYSYTEM


Group 4 Member:
Autsadang Somboonphol
Panisara Kanjanarut
Kornchanok Krajangyao
Phoochit Witchutanon
Taksaorn Aksornsin

# Table of content

# Pain Points

Every time that we go out, the most frustrating is finding a parking spot.

- Even if there is an empty spot, we have to drive around looking for it.
- There is no parking stall, but the security person does not put the "FULL" sign up.
- There is a parking stall, but security does not remove the "FULL" sign.

Group 4

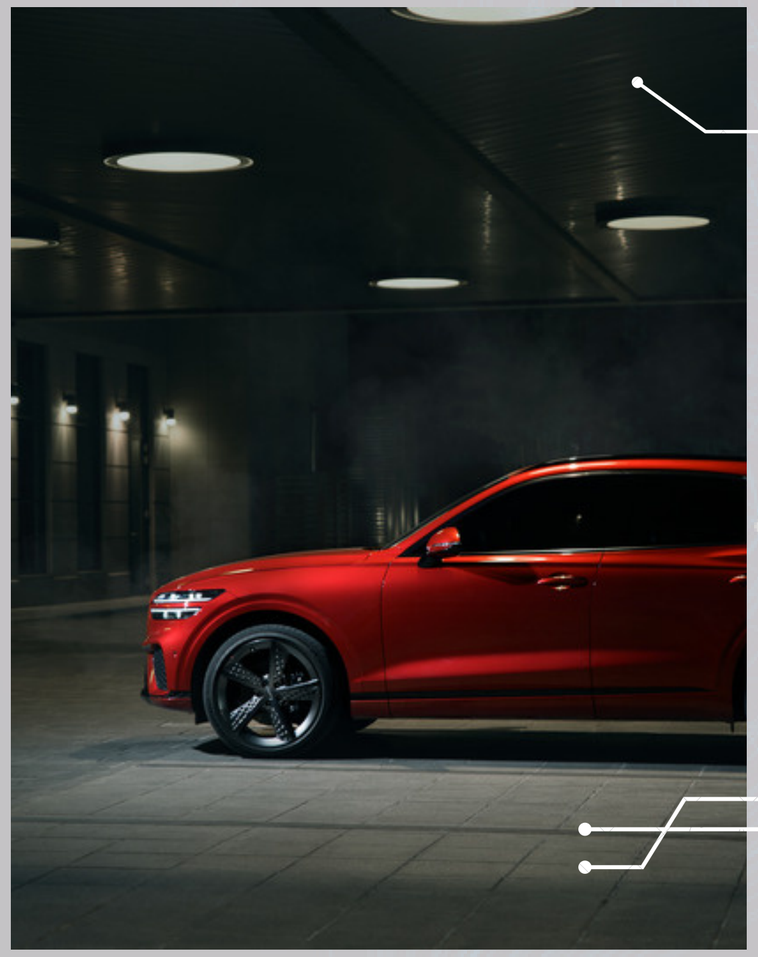
# USER STORIES

# As a Customer

I want to get a parking stall without driving around the parking lot so that I can save time and energy looking for one.

- **Given Parking stall is available, when the car pulls up in front of the gate, then the parking stall number is assigned (Push Style), and the gate opens.**

- **Given FULL parking stall, when the customer will be notified that the parking is full, and the gate will not open.**



# As an owner

I want to see how many cars are in the parking lot so that I can understand the current status.

- Scenario: Density, given parking card are assigned, when checked, then the percentage occupied is displayed.

- Scenario: Income, given parked cars' activity, when checked, then the total income from the beginning of the months up to now and last month are reported.


# As an owner

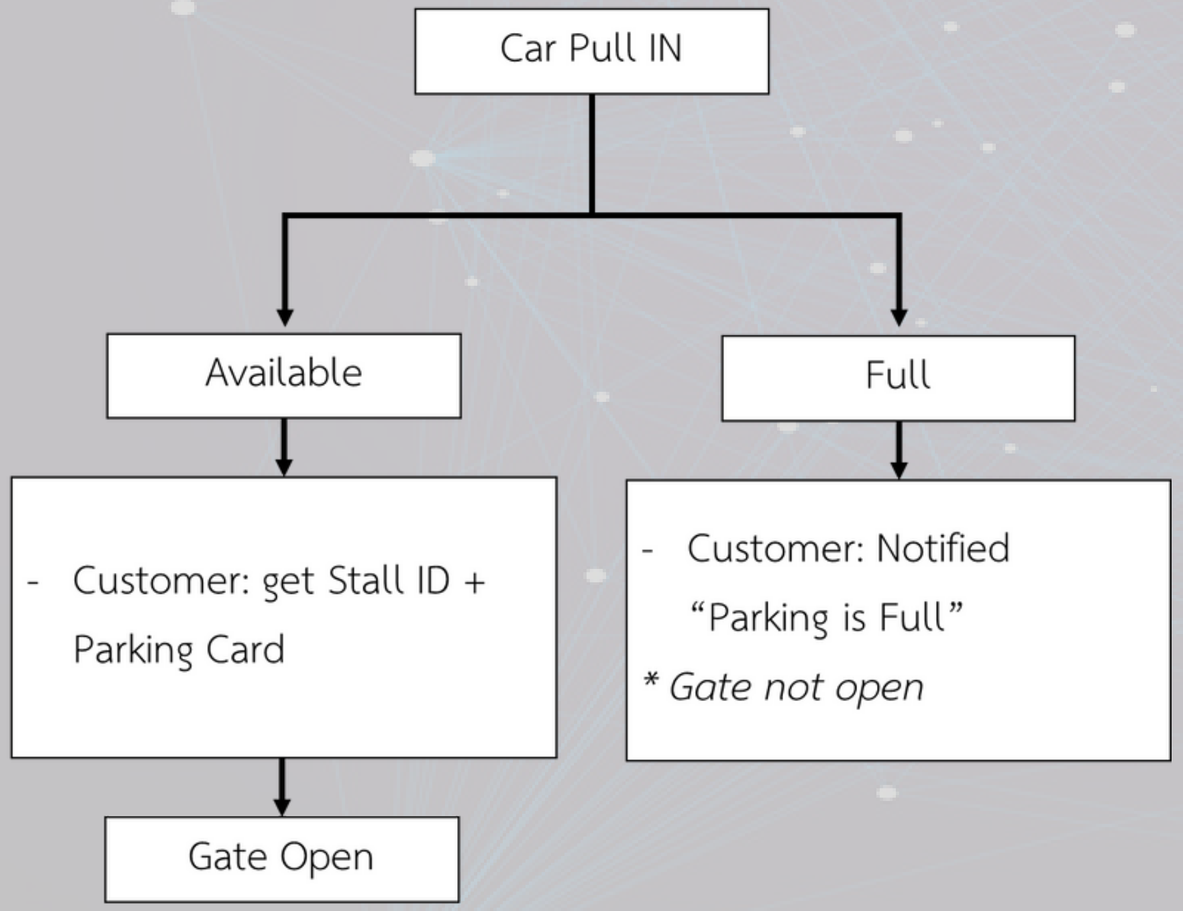
- Scenario: Utilization, given the percentage of stall occupied is calculated, when the number is low, then the space can be allocated for other purposes.

Group 4

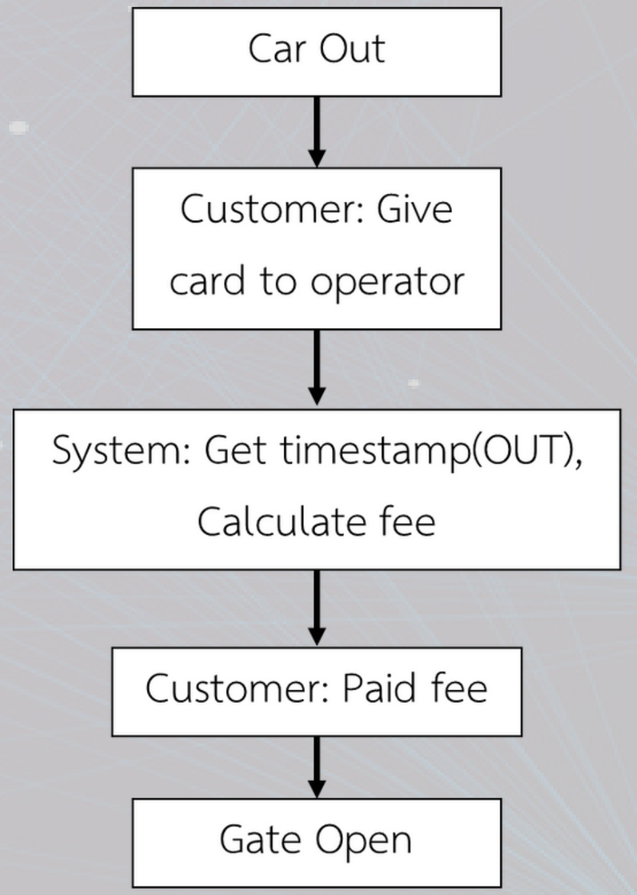
# SYSTEM OVERVIEW

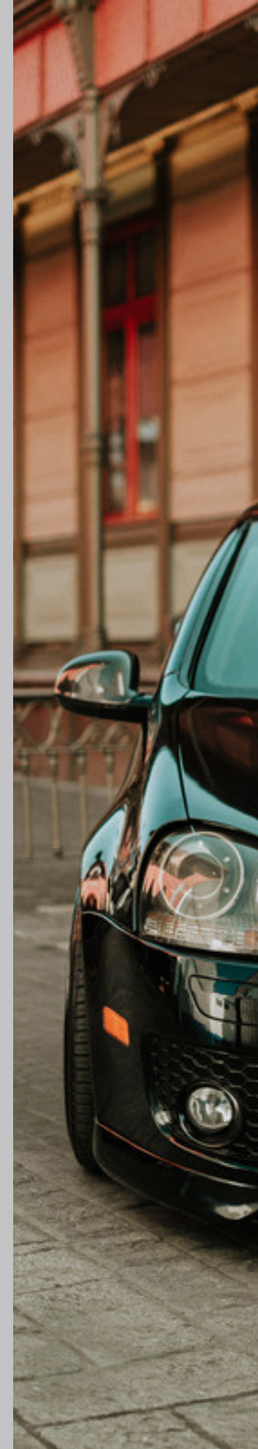
# Customer View

## Scenario: CarIN

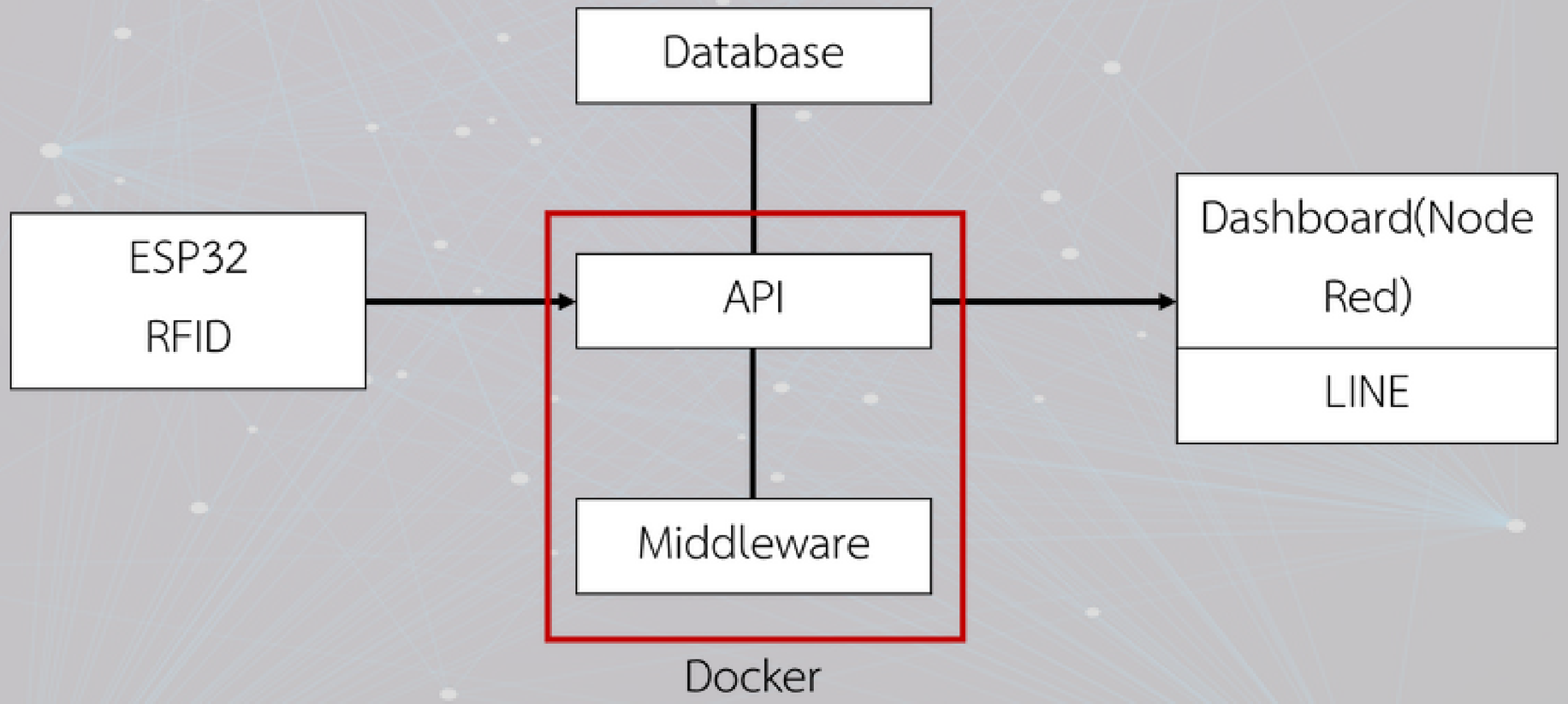



## Scenario: CarOUT

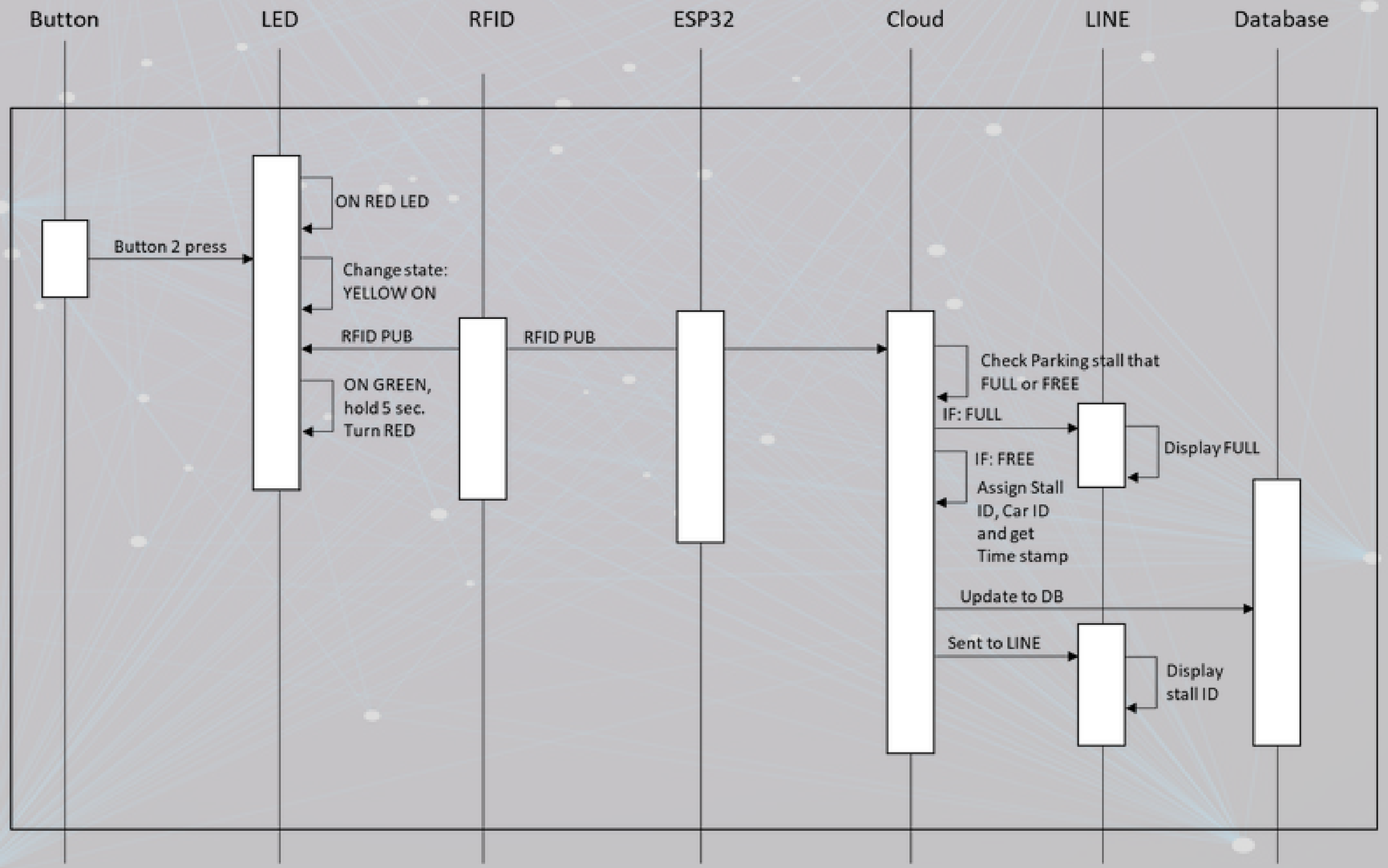


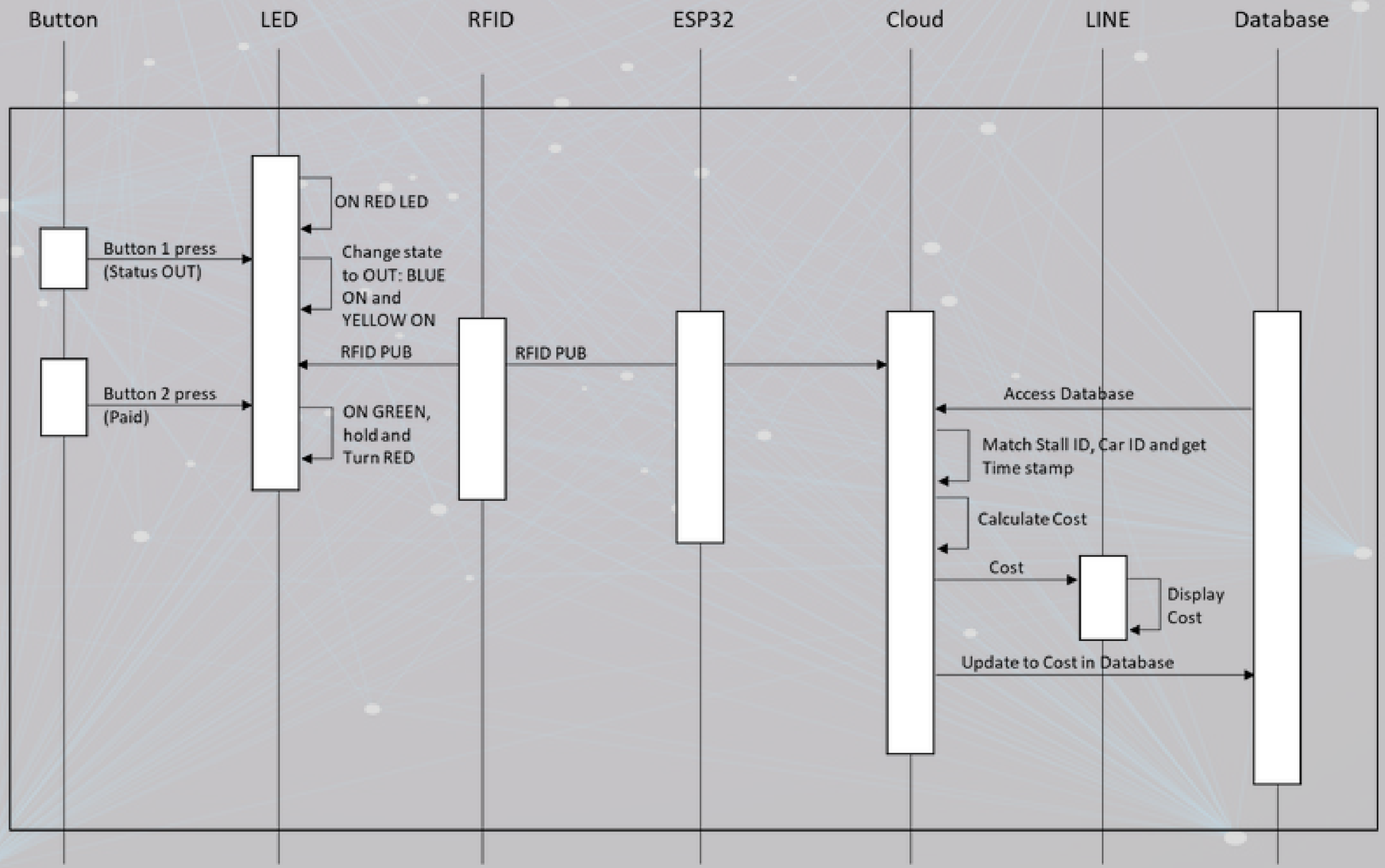
# System Structure



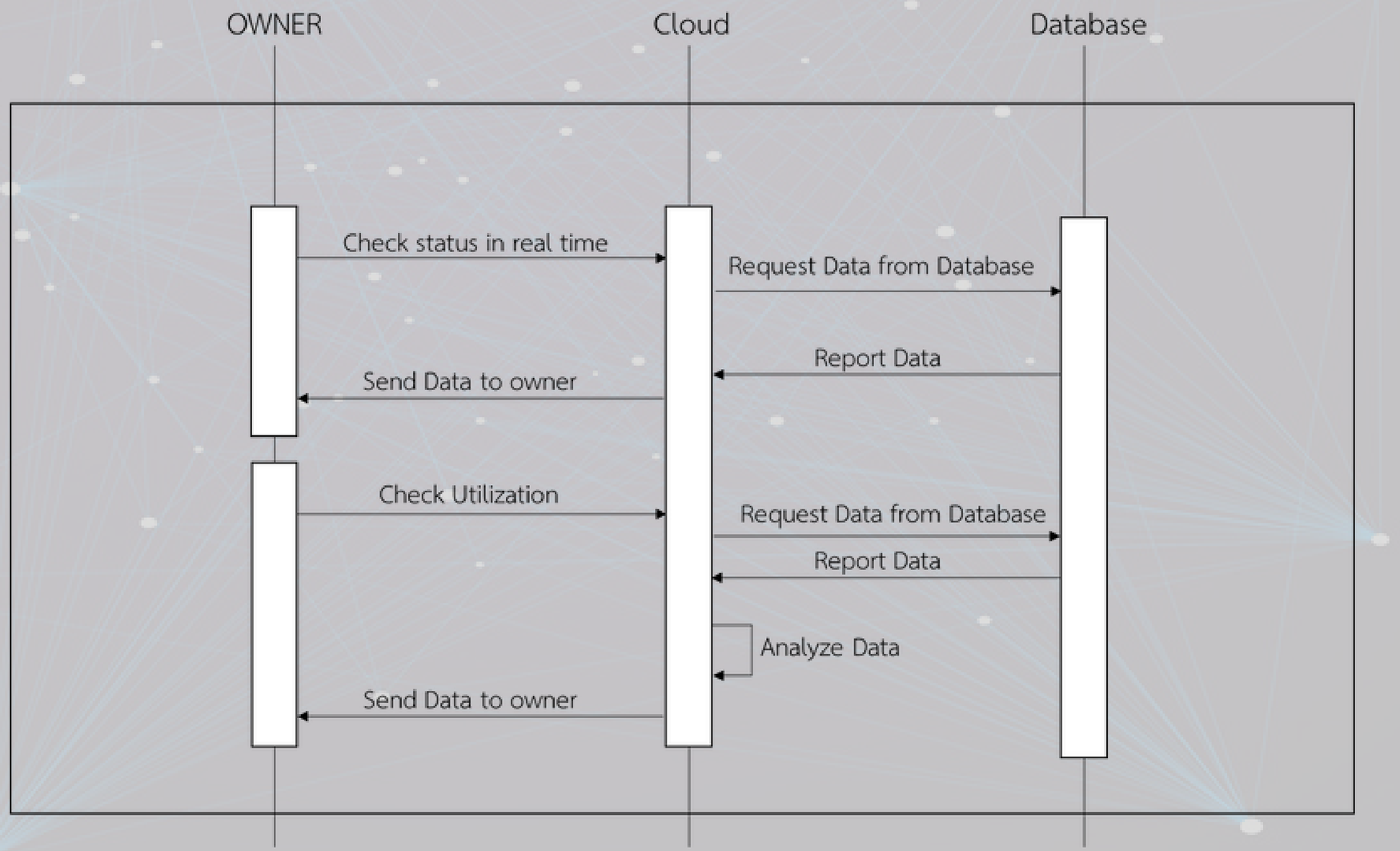
# Sequence: CarIN



# Sequence: CarOUT



# Sequence: Owner



Group 4

# HARDWARE

# The hardware part of the parking system

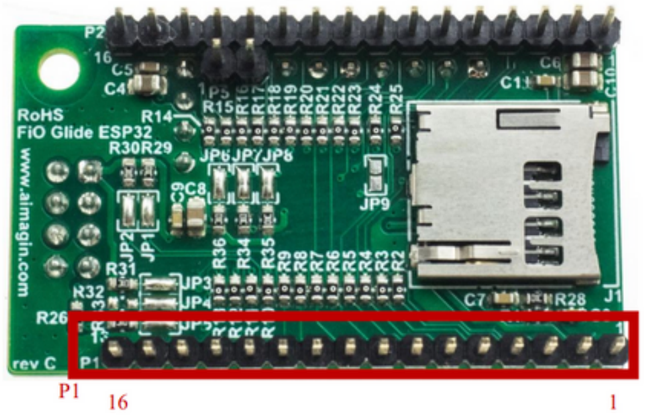
- aMG FiO Glide ESP32
- Adafruit PN532 RFID/NFC Breakout and Shield

# aMG FiO Glide ESP32



## FiO Glide ESP32 Pin Assignment

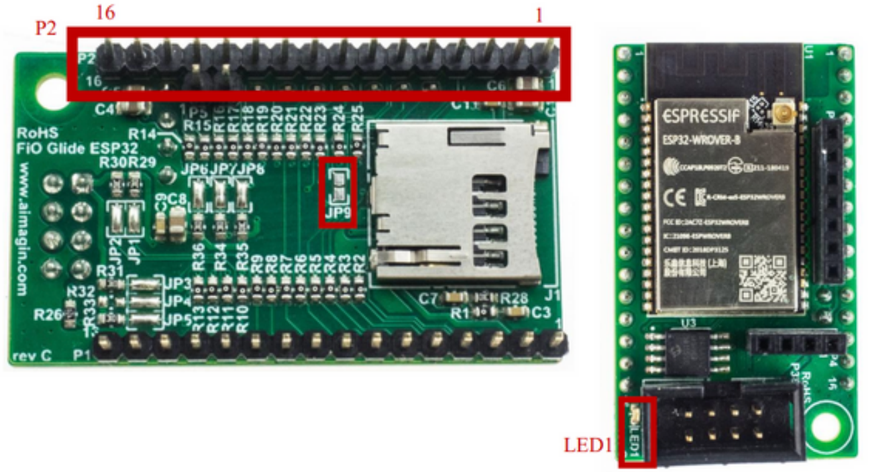
### GPIO - P1




| NO. | Pin/GPIO | Description |
|---|---|---|
| 1 | GND | Ground |
| 2 | 5V | 5V input |
| 3 | D-WDI | |
| 4 | GPIO36 | |
| 5 | GPIO39 | |
| 6 | GPIO34 | |
| 7 | GPIO35 | UART1-RxD |
| 8 | GPIO32 | UART1-TxD |
| 9 | GPIO33 | |
| 10 | GPIO25 | |
| 11 | GPIO26 | I2C-SDA<br>JP1: Default closed to pull-up |
| 12 | GPIO27 | I2C-SCL<br>JP2: Default closed to pull-up |
| 13 | GPIO14 | SPI-SCK<br>JP3: Default closed to pull-up |
| 14 | GPIO12 | SPI-SO<br>JP4: Default closed to pull-up |
| 15 | GPIO13 | SPI-SI<br>JP5: Default closed to pull-up |
| 16 | GND | Ground |

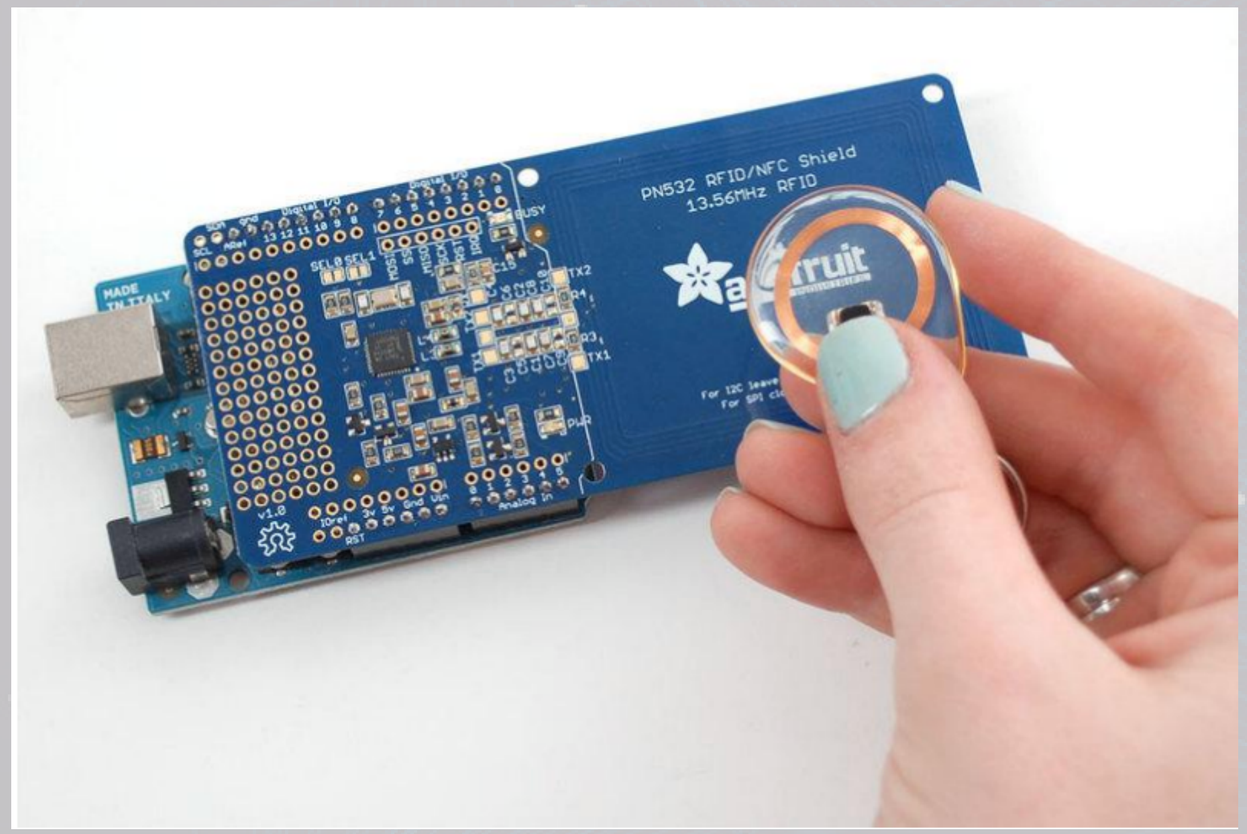
FiO Glide ESP32 (Rev-C) Datasheet

### GPIO - P2




| NO. | Pin/GPIO | Description |
|---|---|---|
| 1 | GND | Ground |
| 2 | 3V3 | +3.3V |
| 3 | EN | Enable |
| 4 | GPIO23 | |
| 5 | GPIO22 | JP9: Default open (Close JP9 for connect GPIO22 to LED1) |
| 6 | GPIO1 | UART0-TxD |
| 7 | GPIO3 | UART0-RxD |
| 8 | GPIO21 | |
| 9 | GPIO19 | |
| 10 | GPIO18 | |
| 11 | GPIO5 | |
| 12 | GPIO4 | |
| 13 | GPIO0 | JP8: Default closed to pull-up |
| 14 | GPIO2 | JP7: Default closed to pull-down |
| 15 | GPIO15 | SPI-CS<br>JP6: Default closed to pull-up |
| 16 | GND | Ground |

# Adafruit PN532 RFID/NFC Breakout and Shield



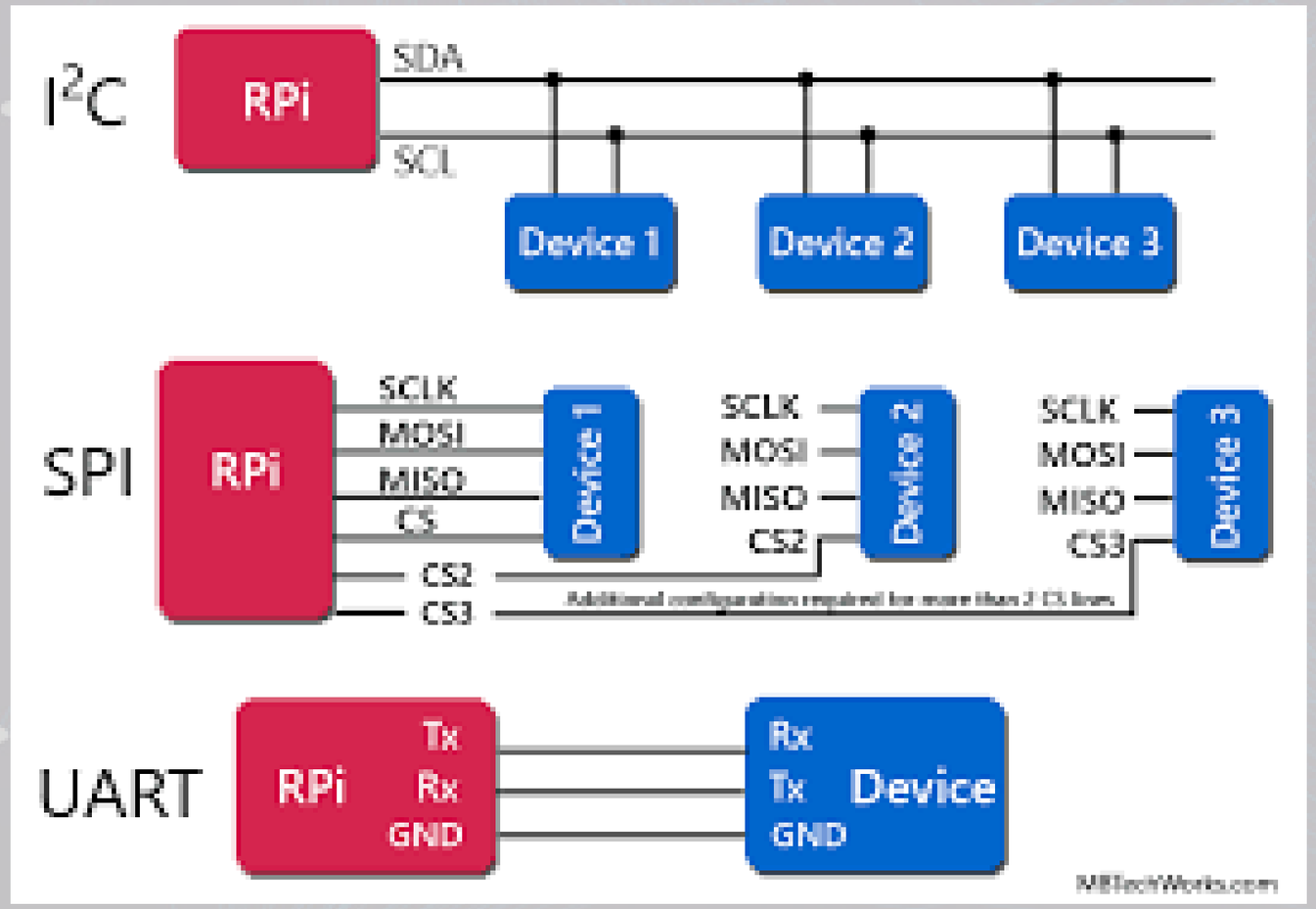

Conection

- SPI
- TTL(TX/RX)
- I2C

# I2C vs SPI

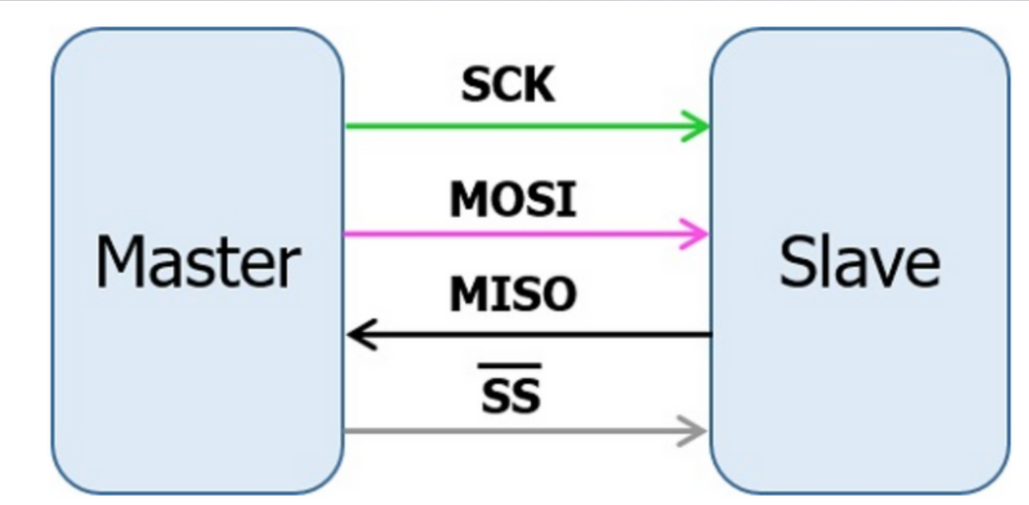
- I2C is half-duplex communication, and SPI is full-duplex communication.
- I2C supports multi-master and multi-slave and SPI supports single-master.
- I2C is a two-wire protocol, and SPI is a four-wire protocol.
- I2C supports clock stretching, and SPI does not have clock stretching.
- I2C is slower than SPI.
- I2C has extra overhead start and stop bits, and SPI does not have any start and stop bits.
- I2C has an acknowledgment bit after every byte of transfer.
- I2C has a pullup resistor requirement.

# I2C vs SPI vs TTL




TTL stands for Transistor-Transistor Logic, a serial communication commonly found in UART (universally asynchronous receiver/transmitter)

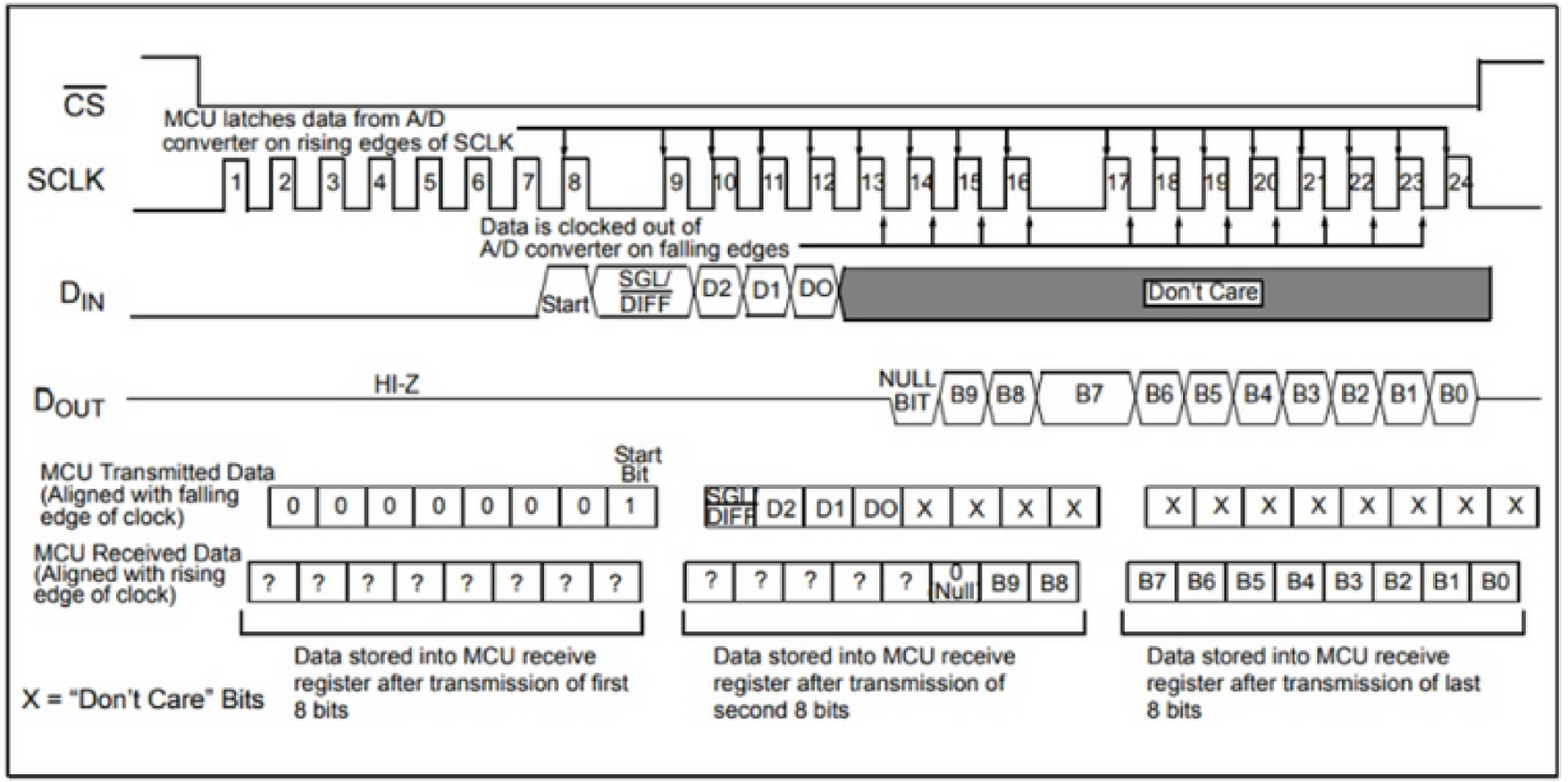
# SPI




The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication. The SPI bus specifies four logic signals:

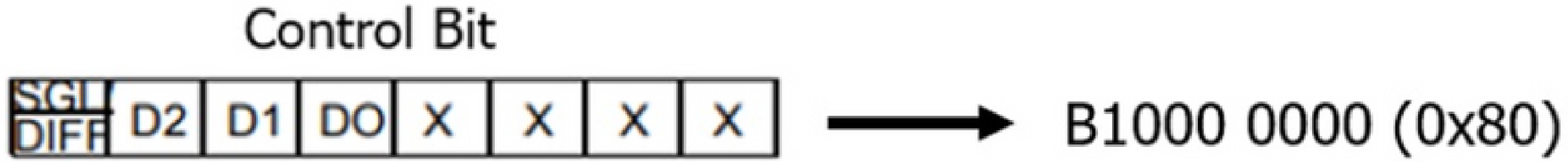
- SCLK: Serial Clock (output from master)
- MOSI: Master Out Slave In (data output from master)
- MISO: Master In Slave Out (data output from slave)
- CS /SS: Chip/Slave Select (often active low, output from master to indicate that data is being sent)

# SPI




SPI Communication with the MCP3004/3008 using 8-bit segments

# SPI



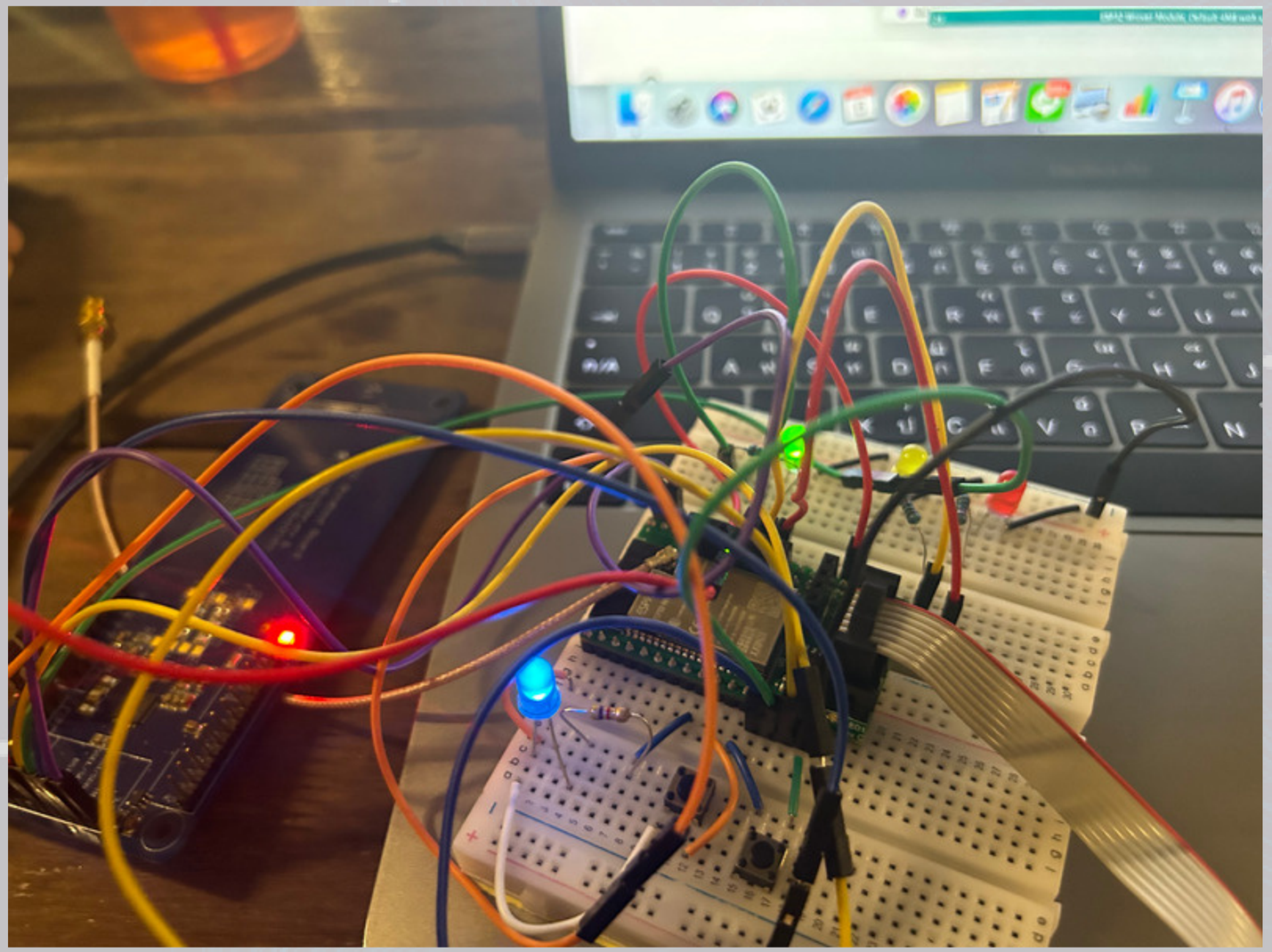

```
if (uidLength == 4)
{
  // We probably have a Mifare Classic card ...
  uint32_t cardid = uid[0];
  cardid <<= 8;
  cardid |= uid[1];
  cardid <<= 8;
  cardid |= uid[2];
  cardid <<= 8;
  cardid |= uid[3];
  Serial.print("Seems to be a Mifare Classic card #");
  Serial.println(cardid);
```

<<= is left shift bits of set by 8 (a self-assigned form of << bitwise left shift operator)

|= reads the same way as +=

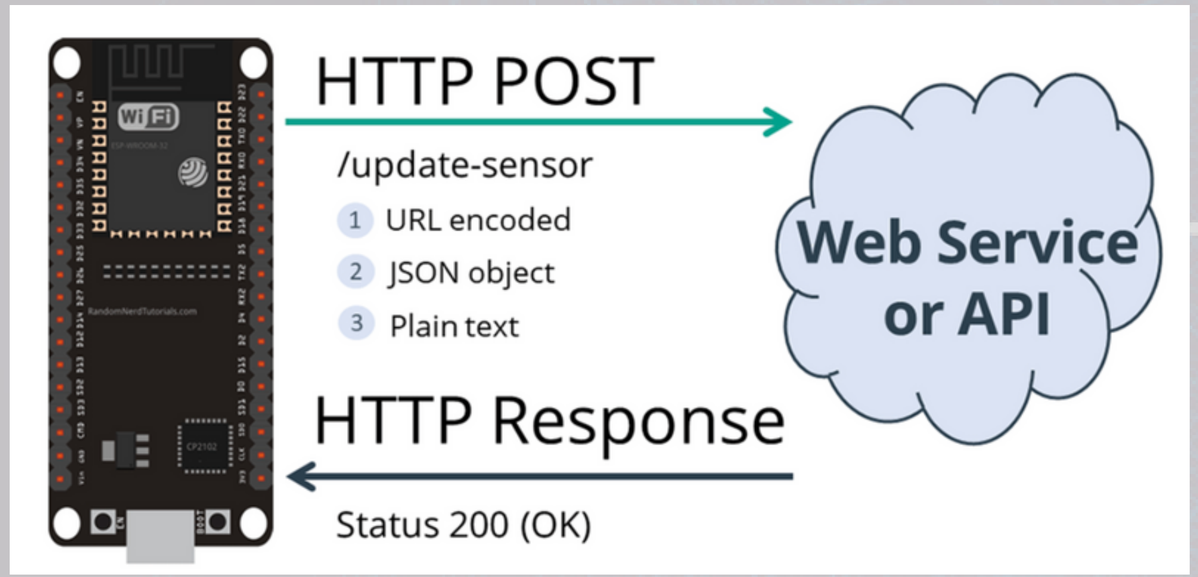
where | is the bit-wise OR operator.

# Pin connection



```
#define BUTTON_PIN_1 26
#define BUTTON_PIN_2 27
#define LED_PIN 33
#define LED_R_1 18
#define LED_Y_1 19
#define LED_G_1 21
#define PN532_SCK  (14)
#define PN532_MOSI (13)
#define PN532_SS   (15)
#define PN532_MISO (12)
```

# HTTP POST

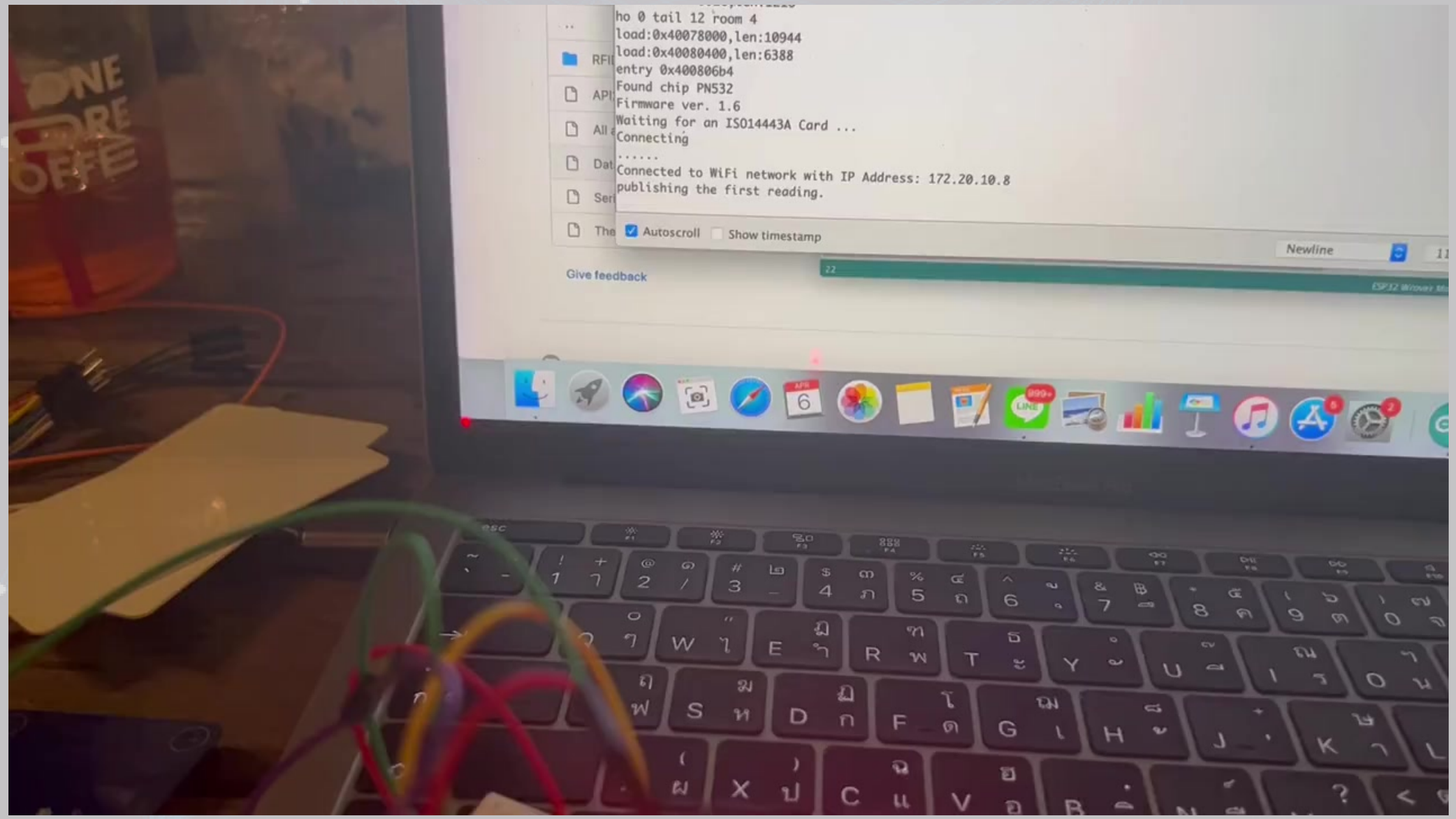



```
.....
Connected to WiFi network with IP Address: 172
publishing the first reading.
Found an ISO14443A card
  UID Length: 4 bytes
  UID Value: 0x34 0xDD 0xE2 0x2B
Seems to be a Mifare Classic card #886956587
HTTP Response code: 200

Found an ISO14443A card
  UID Length: 4 bytes
  UID Value: 0x3B 0x82 0xB2 0xA4
Seems to be a Mifare Classic card #998421156
HTTP Response code: 200
```

{Car_id: 886956587 }

# Demonstration



Group 4

# BACKEND

# Backend system Summary

- We use python Fastapi as a RestAPI  system.
- We use MongoDB  as a system database.
- We use Docker as a platform to deployment

# Database

- We use MongoDB as database the database contain  3 collection

**Car_in**

```
{
        _id :<Objectid1>,
        time _in : "2023-03-01
12:03:06"
}
```

**Car_out**

```
{
        '_id': <Objectid1>,
        'car_id': 'ABC123',
        'time_in':  '2021-05-01 12:00:00',
        'time_out': '2021-05-01 13:00:00',
        'stall_id': 'A',
         'fee': 20
}
```

**parking**

```
{
        '_id': <Objectid1>,
        'stall_id': 'A',
        'stall_status':1
        'car_id':'ABC123'
}
```

# REST-API

## Implement with FastAPI

On start up
Initialize the database if not exist

- define database collection
- define column names for each collection

Open tunnel by **ngrok**

```python
@app.on_event("startup")
def startup_db_client():
    print("Connected to the MongoDB database!")

    ngrok.set_auth_token(os.environ['NGROK_TOKEN'])
    public_url = ngrok.connect(8000)
    url = public_url.public_url.replace('http', 'https') + '/callback'
    print(url)
    dblist = mongodb_client.list_database_names()
    if "parking_db" in dblist:
        print("The database exists.")
    else:
        print("The database does not exist.")
        print("The database and collections is now being created ... ")
        db = mongodb_client["parking_db"]
        car_in = db[os.environ['CAR_IN']]
        car_out = db[os.environ['CAR_OUT']]
        parking = db[os.environ['PARKING']]
        parking_list = ["A","B","C","D"]
        for i in parking_list:
            parking.insert_one(
                {
                    "stall_id":i,
                    "stall_status":0,
                    "car_id":None}
                )
        print("The database and collections have been created.")
```

# REST-API

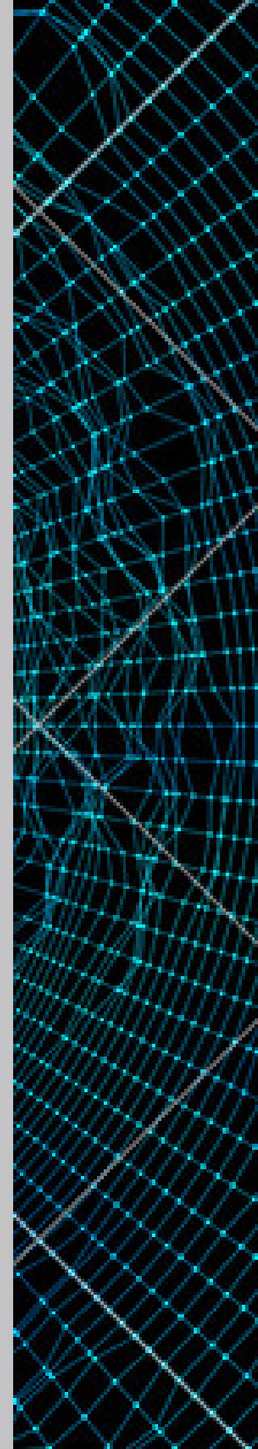
## Implement with FastAPI

**On start up -** Initialize the database if not exist

- define database collection
- define column names for each collection

```
@app.on_event("shutdown")
def shutdown_db_client():
    mongodb_client.close()
    print("MongoDB connection closed.")
```

**On shut down** - Close the mongo db



# REST-API

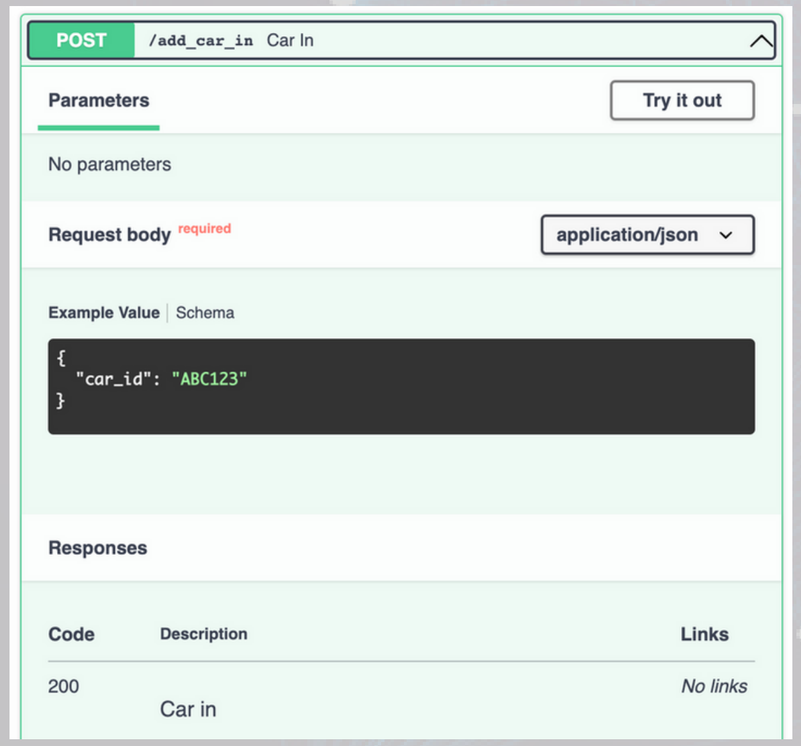
This is a api route for this system

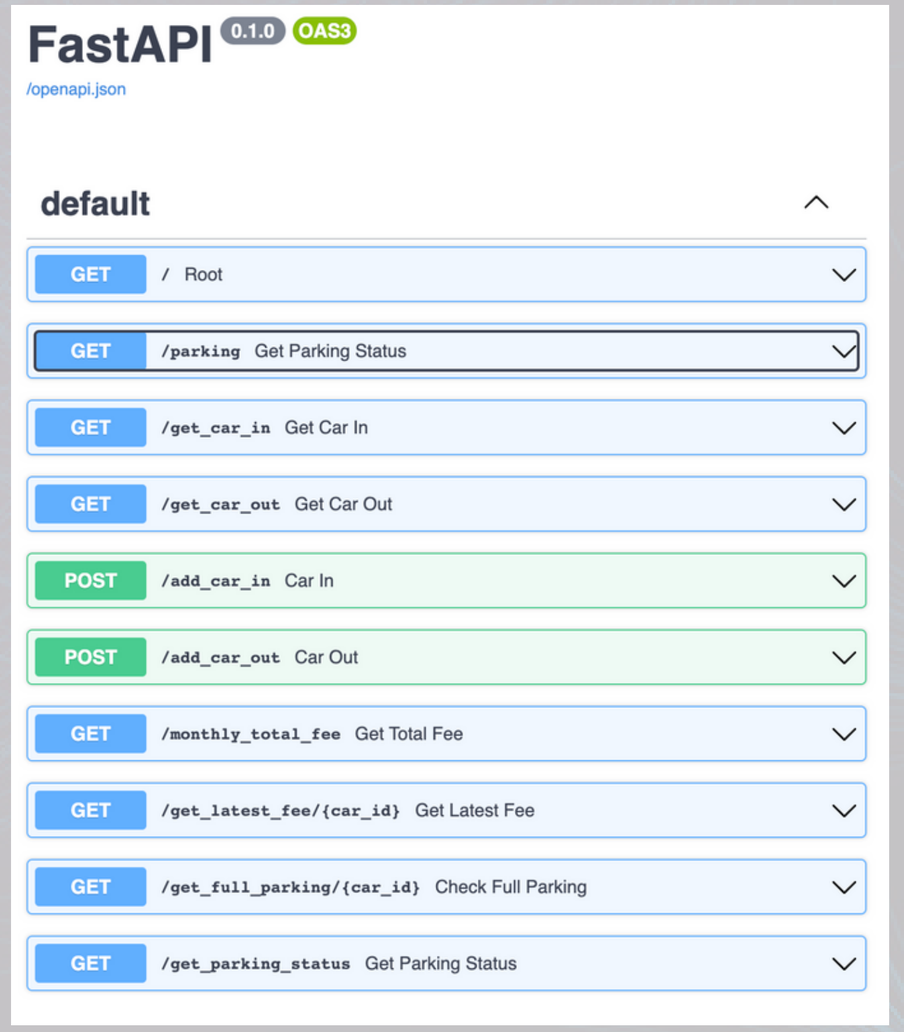
GET/ Root
GET/Parking_Status
GET/get_car_in
GET/get_car_out
GET/monthly_total_fee
GET/get_latest_fee/{car_id}
GET/get_full_parking/{car_id}

POST/add_car_in
POST/add_car_out

# REST-API

include the example



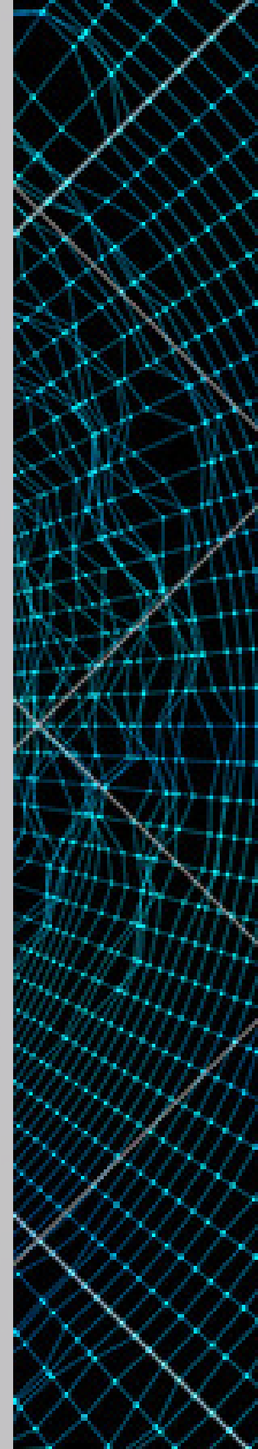



# REST-API

This is a api route for this system

**GET**/ Root
**GET**/Parking_Status
**GET**/get_car_in
**GET**/get_car_out
**GET**/monthly_total_fee
**GET**/get_latest_fee/{car_id}
**GET**/get_full_parking/{car_id}

**POST**/add_car_in
**POST**/add_car_out



# REST-API

POST add_car_in

POST add_car_out

Added the condition for Check that car already exist or not

```python
@app.post("/add_car_in", response_description="Car in")
def car_in(car_in: Car_in):
    car_in = jsonable_encoder(car_in)
    # get current time as YYYY-MM-DD HH:MM:SS
    current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    # check if parking is full
    if not db[os.environ["PARKING"]].find_one({"stall_status":0}):

        message = "Parking is full"
        requests.post(line_url, headers=line_headers, data = {'message':message})

        raise HTTPException(status_code=400, detail="Parking is full")

    elif not db[os.environ["PARKING"]].find_one({"car_id":car_in["car_id"]}):
        assign_parking = db[os.environ["PARKING"]].find_one({"stall_status":0})
        car_in_insert_data = {
            "time_in":current_time,
            "stall_id":assign_parking["stall_id"],
            "car_id":car_in["car_id"]
            }
        parking_update_data = {
            "stall_status":1,
            "car_id":car_in["car_id"]
            }
        db[os.environ["CAR_IN"]].insert_one(car_in_insert_data)
        db[os.environ["PARKING"]].update_one(
            {"stall_id":assign_parking["stall_id"]},
            {"$set":parking_update_data})
        message = f"Car : {car_in} inserted into {assign_parking['stall_id']}"

        requests.post(line_url, headers=line_headers, data = {'message':message})

        return {"message": f"Car : {car_in} inserted into {assign_parking['stall_id']}"}

    else :
        raise HTTPException(status_code=400, detail="Car already in the parking")
```

# REST-API

Design Model for check name  the key value of .json

```python
class Car_in(BaseModel):
    car_id: str = Field( ... )
    class Config:
        allow_population_by_field_name = True
        schema_extra = {
            "example": {
                "car_id": "ABC123"
            }
        }

class Car_out(BaseModel):
    car_id: str = Field( ... )

    class Config:
        allow_population_by_field_name = True
        schema_extra = {
            "example": {
                "car_id": "ABC123"
            }
        }
```

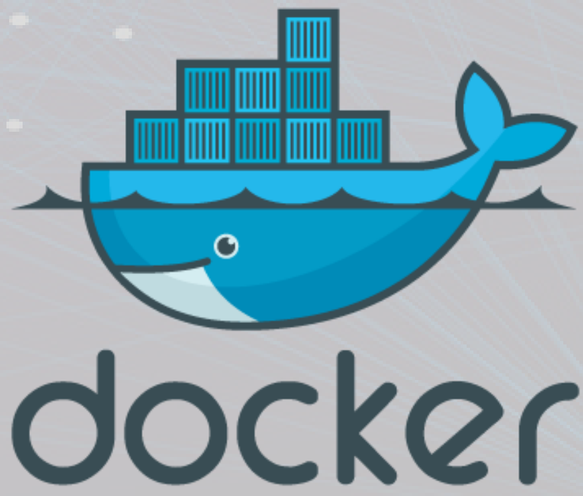
```yaml
# ref: https://docs.docker.com/compose/compose-file/
version: '3.8'
services:
  api:
    container_name: api_parking
    build:
      context: ./api
    ports:
      - "8000:8000"
    volumes:
        - ./api:/workspace
    env_file:
        - .env
    depends_on:
    - mongo_db
    restart: always

  mongo_db:
    container_name: parking_mongo
    image: mongo
    restart: always
    ports:
      - "27017:27017"
  node_red:
    container_name: node_red
    image: nodered/node-red
    ports:
      - "1880:1880"
    volumes:
      - ./node_red:/data
      - ./dashboard/:/data/
    command:
      - echo "node-red-dashboard"
    restart: always
  linenotify:
    container_name: linenotify
    build: ./Linebot
    ports:
      - "8888:8888"
    env_file:
      - .env
    restart: always
```

# Docker

We deploy system with Dockerfile and docker-compose. This docker have 3 container.

- api_parking
- node_red
- mongo_db



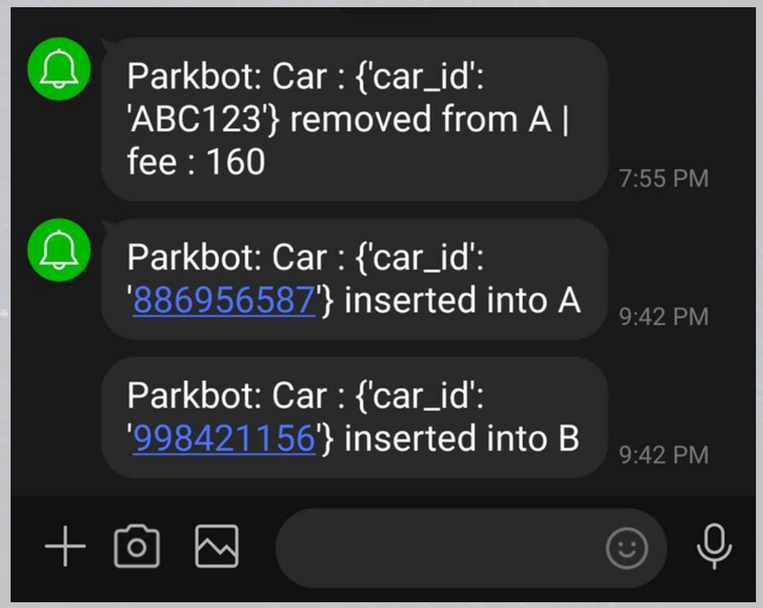
Group 4

# FRONTEND

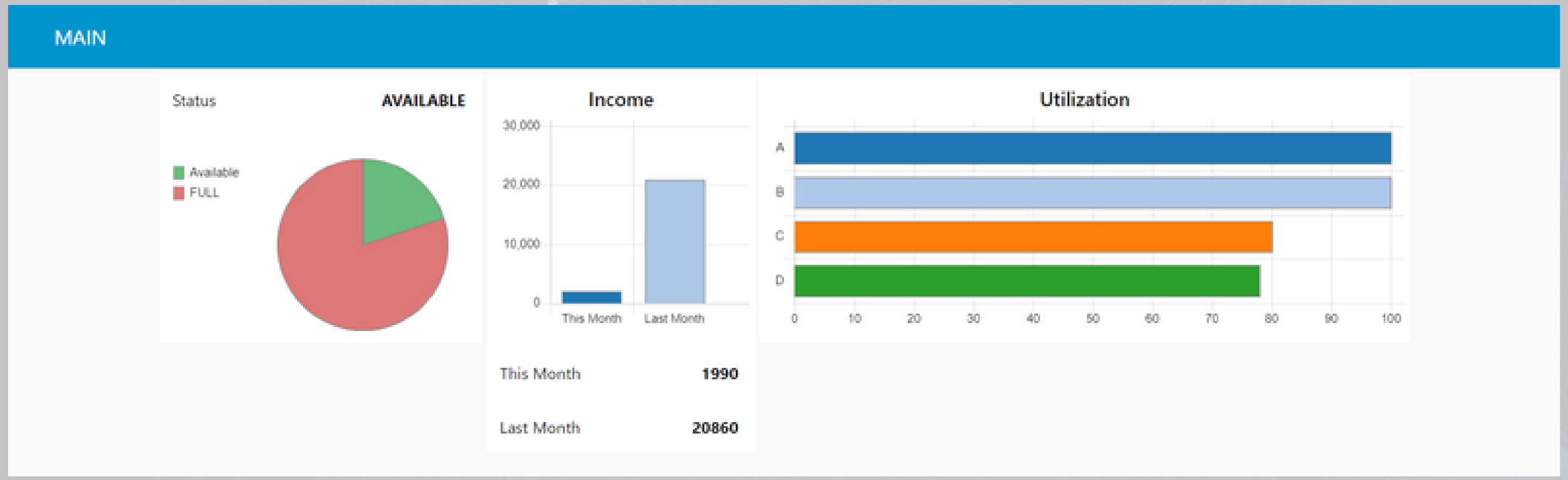
# Line notify

In order to notify user ,we provided notification about

- Welcome to Parking System.
- Check parking stalls are full or not?
- Inform Stall id  to parked
- Inform parking fee to user



# Dashboard




- Give quick overview of the system for owner
- still under implementations

Group 4

# POTENTIAL BENEFITS

Group 4

# Operation Management

Given there are databases of customer behavior

when the statistic of customer behavior is determined,

then the department can adjust their operation like

- Opening/closing time
- Employees hired
- Facility investment accordingly

Group 4

# Advance Booking

Given the statistic of stall occupied is determined,

when the customer wants to know space availability in advance,

then the owner can tell whether there will be available space or not.

This could be another income source or VIP service

Borcelle

# THANK YOU

Until Next Time

Borcelle

# Line notify

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

Borcelle

# Line notify

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

Borcelle

# How it Works?

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

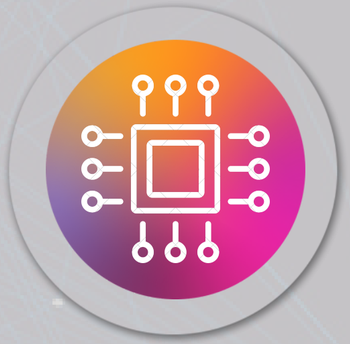
Borcelle

# The Process

Step 1

Step 2

Step 3

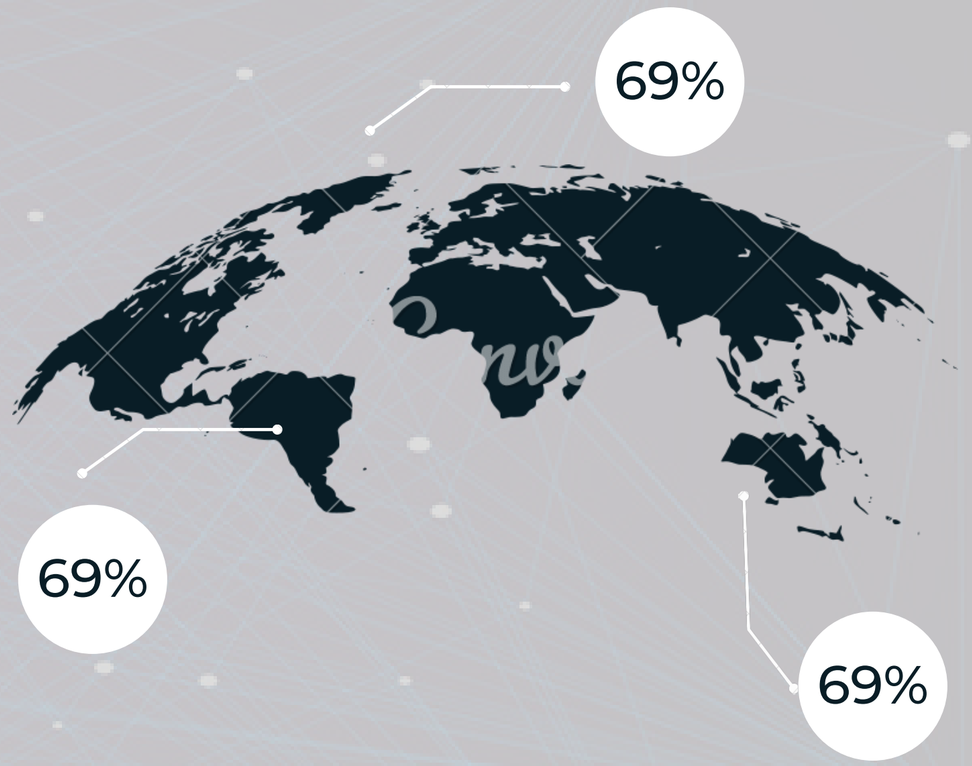


Step 4

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

Borcelle




# Survey

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

Borcelle




# Our Gallery

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

Borcelle

# Our Team

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

Borcelle

# Our Team

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.

Borcelle

# Contact

Presentation are communication tools that can be used to as demonstration, lecture, reports, and more.


123-456-7890

www.reallygreatsite.com

Hello@reallygreatsite.com

123 Anywhere St., Any City, ST 12345