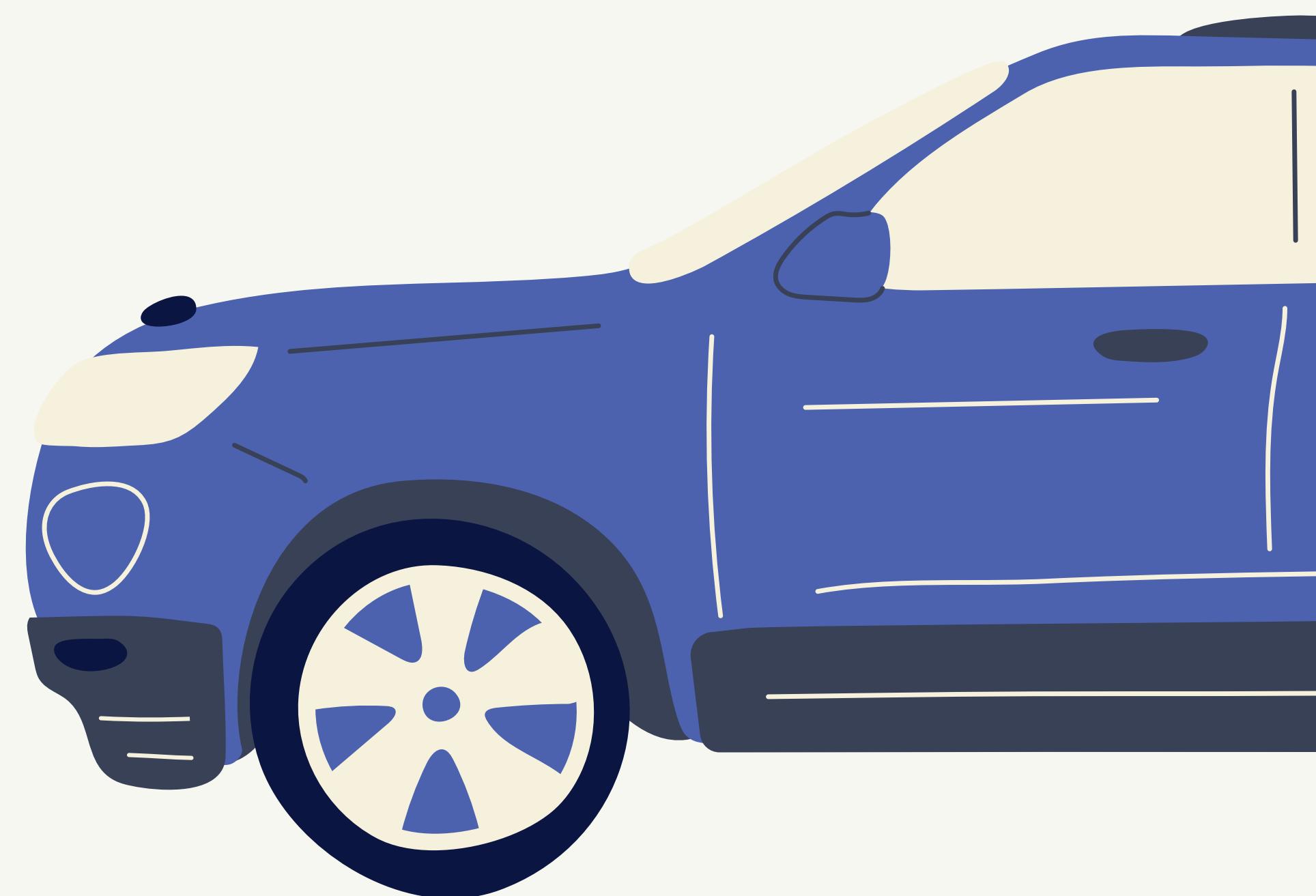


# AN APPLICATION FOR HOME-TO-PUBLIC PARKING

แอปพลิเคชันสำหรับการเปลี่ยนบ้านเป็น  
ที่จอดรถสาธารณะ

CE66-14 Computer Engineering  
King Mongkut's Institute of Technology Ladkrabang



# ADVISOR



Asst. Prof. Dr. Chutimet Srinilta

Department of Computer Engineering, KMITL

# MEMBERS



Kritsanaphat Phanjaroen  
63010040



Kueakun Niyomsit  
63010095

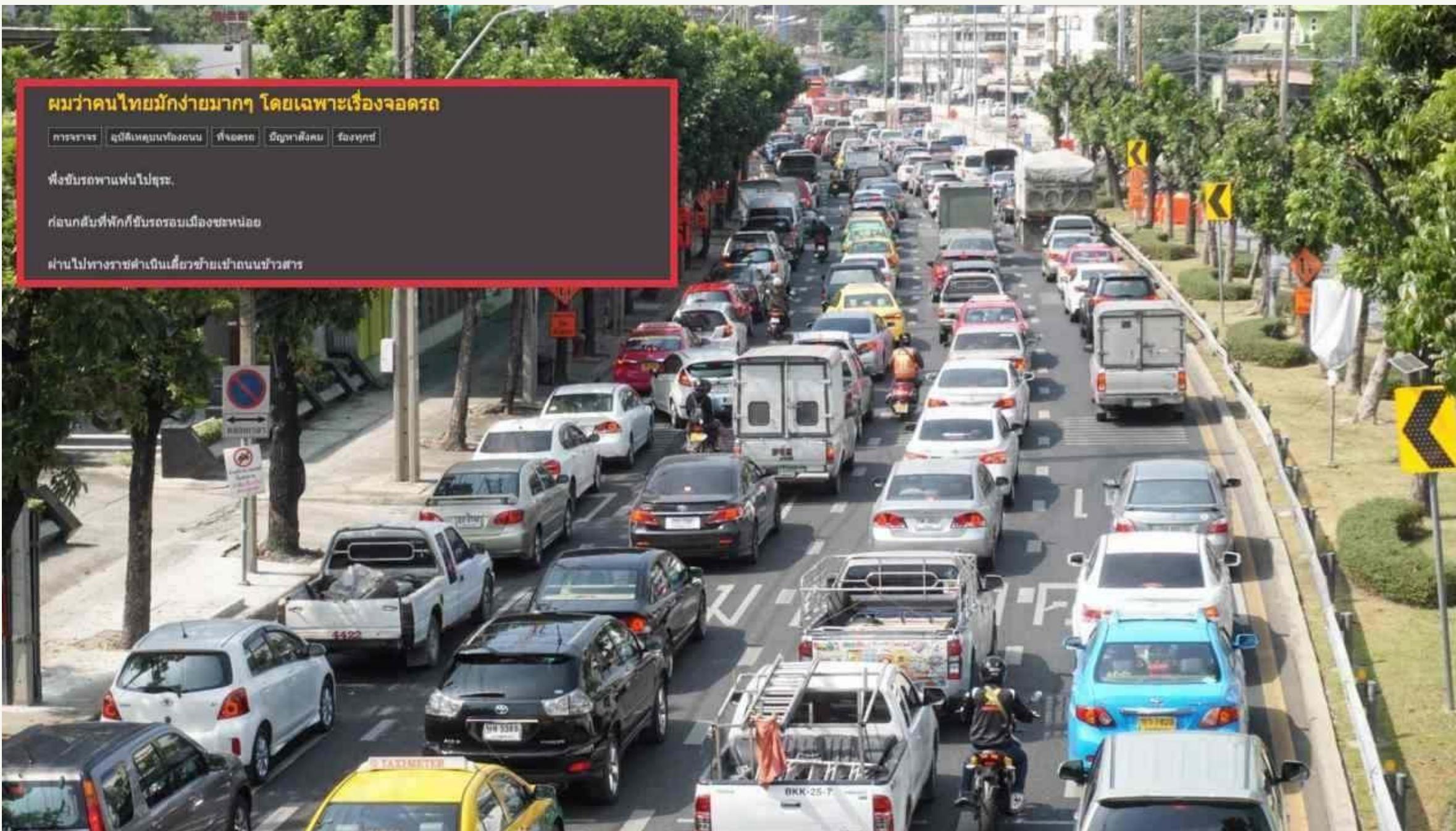


Watcharapol Yotadee  
63010870



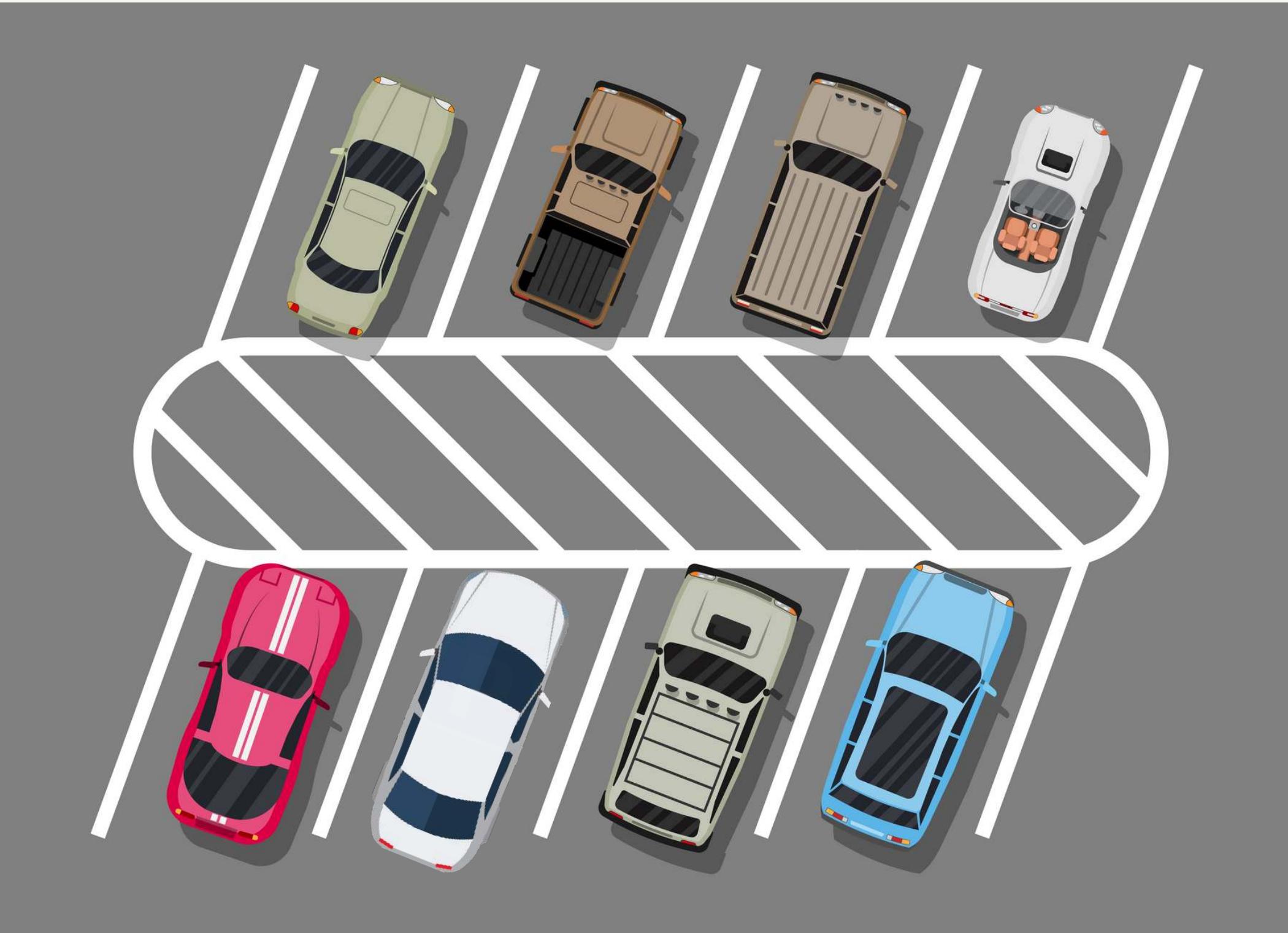
# Introduction

# Introduction



# Problem

จอดไหนดีไม่มีที่  
ว่างเลย



## ประเด็น



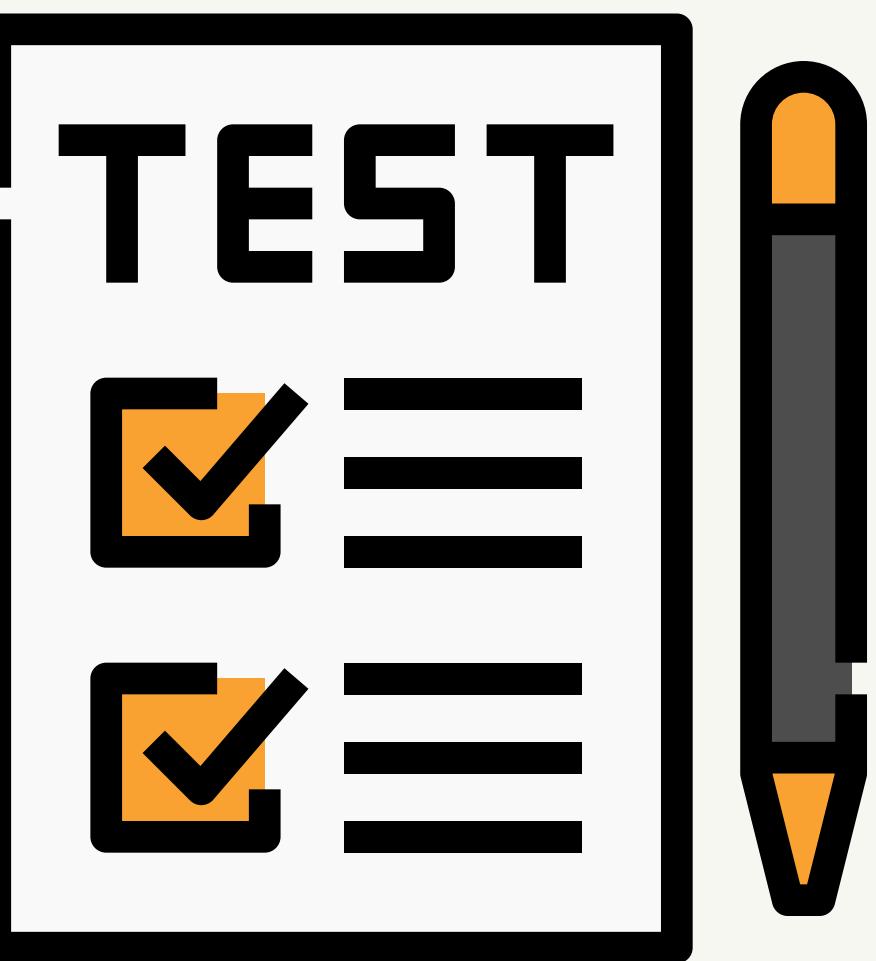
เดี่ยวไปเจอกันที่  
ห้างเดิมนะ ครับ







# น้ำฝน



รถก็ติด

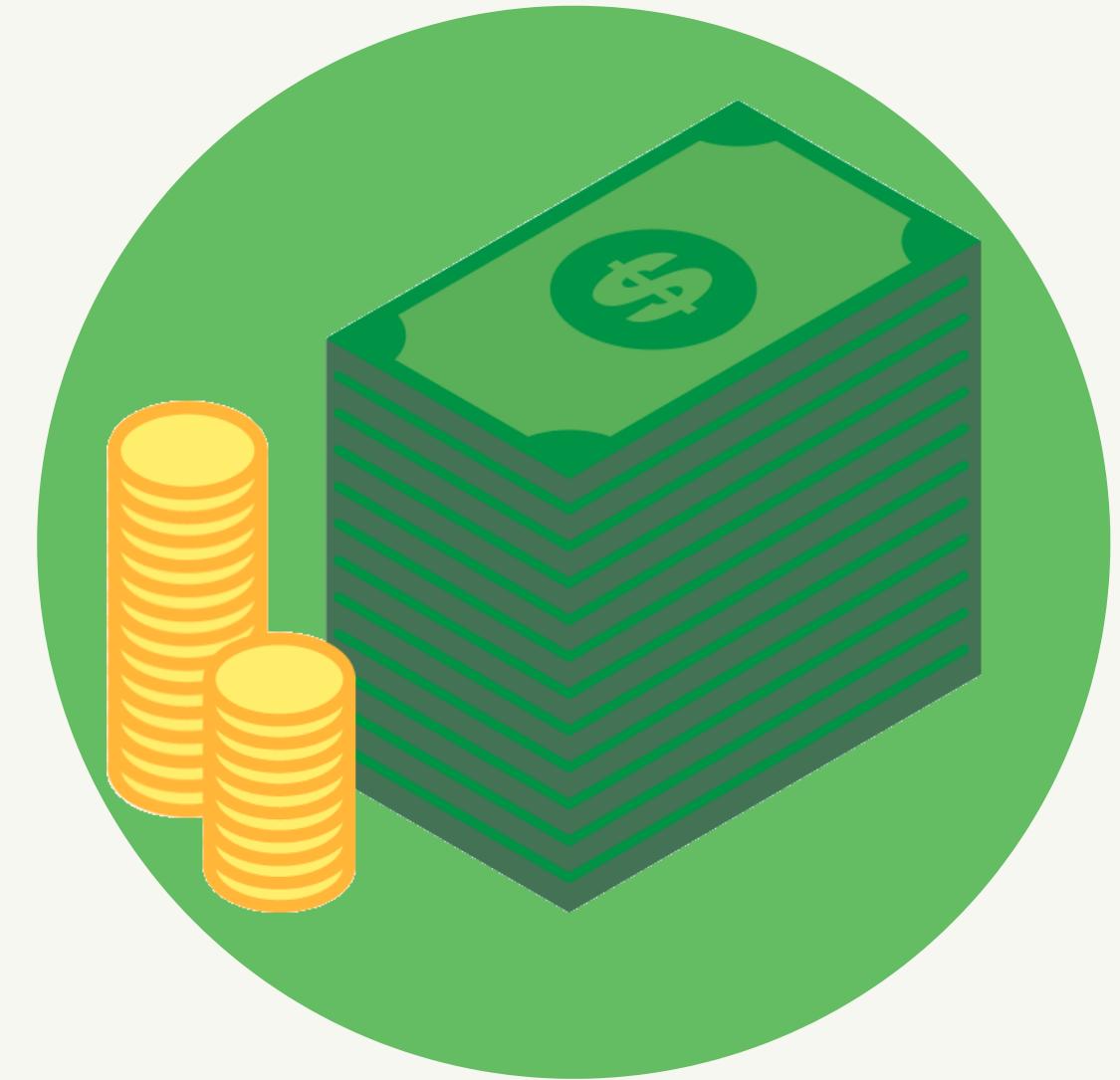
ที่จอดรถก็ไม่มี

สายแน่เลย  
!!!





# Solution





# PARKFINDER

---







# เมื่อเทียบกับ คู่แข่ง



มีระบบให้จองล่วงหน้าเกิน 3 ชั่วโมง



เจ้าของที่จอดลงทุนต่ำ



สามารถตรวจสอบรถได้ตลอดเวลา



focus ที่จอดที่เป็นบ้านคน



ในระบบปกติ จะไม่ใช้คนในการทำงาน



# TechStack

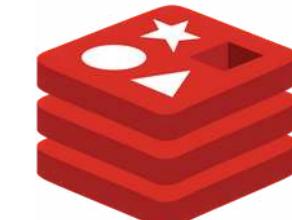
## Frontend



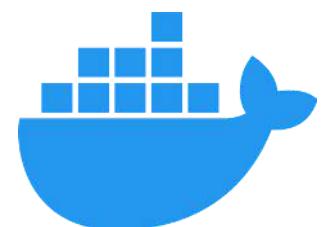
## Backend



## Database



## Services



# Feature



## รถของฉัน

สามารถเพิ่ม แก้ไข และลบรายละเอียดรถได้



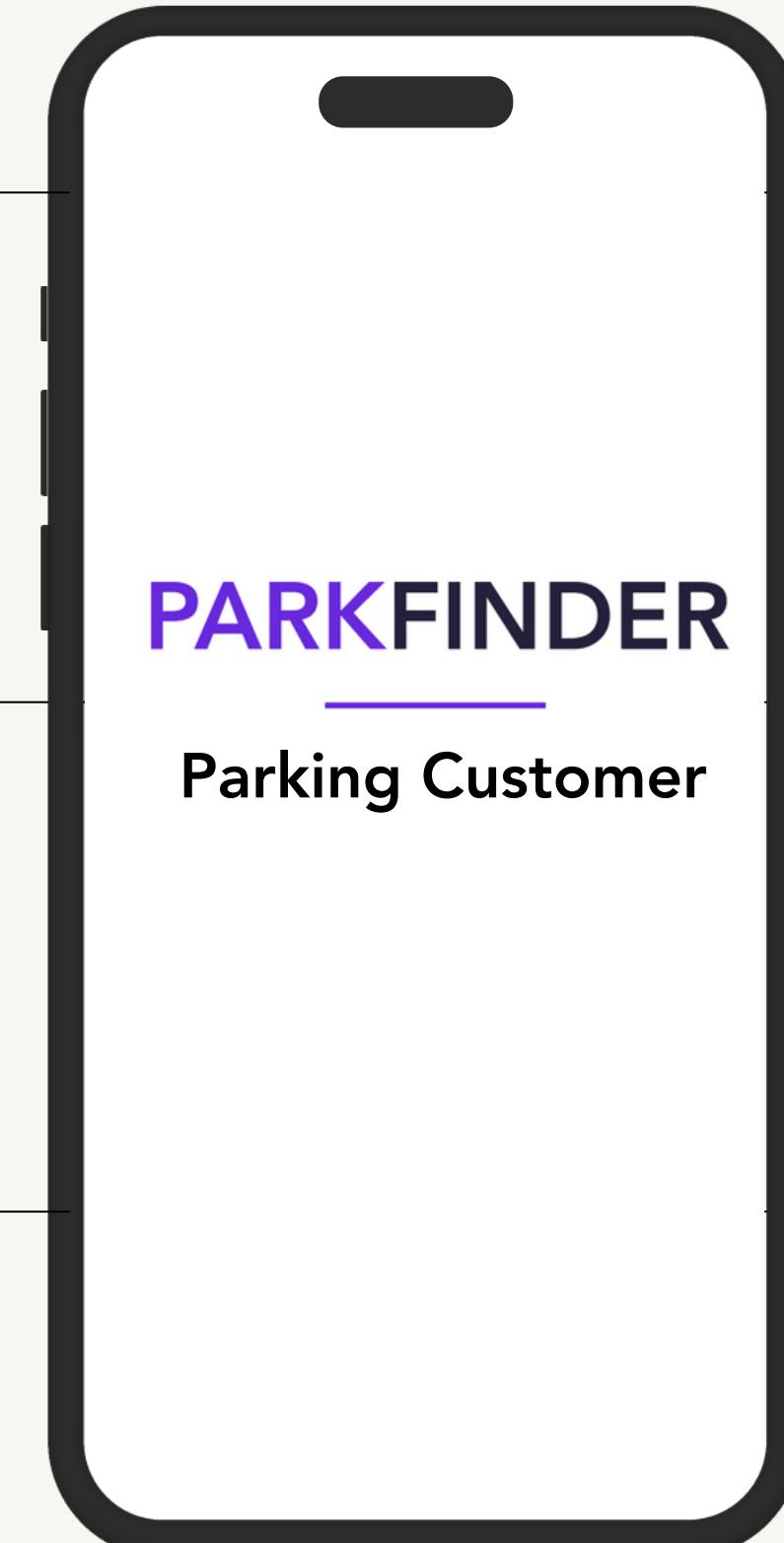
## จองที่จอดรถ

จองหรือเช่าที่จอดรถที่ต้องการ โดยสามารถเลือกวัน - เวลา และจำนวนชั่วโมงที่ต้องการได้



## จองล่วงหน้า

จองที่จอดรถล่วงหน้าได้ ไม่เกิน 3 เดือน



## ขอดูรูป

ขณะจอดรถ สามารถขอดูรูปได้



## ส่งข้อความหาเจ้าของที่

ขณะจอดรถ สามารถส่งข้อความหาเจ้าของที่จอดรถได้



## แลกรewards

สามารถดู และแลกรewards ได้

# Feature



## ที่จอดรถของฉัน

สามารถเพิ่ม แก้ไข และลบรายละเอียดที่จอดรถได้



## ขอดูรูปรถ

สามารถขอดูรูปถ่ายที่จอดอยู่ได้



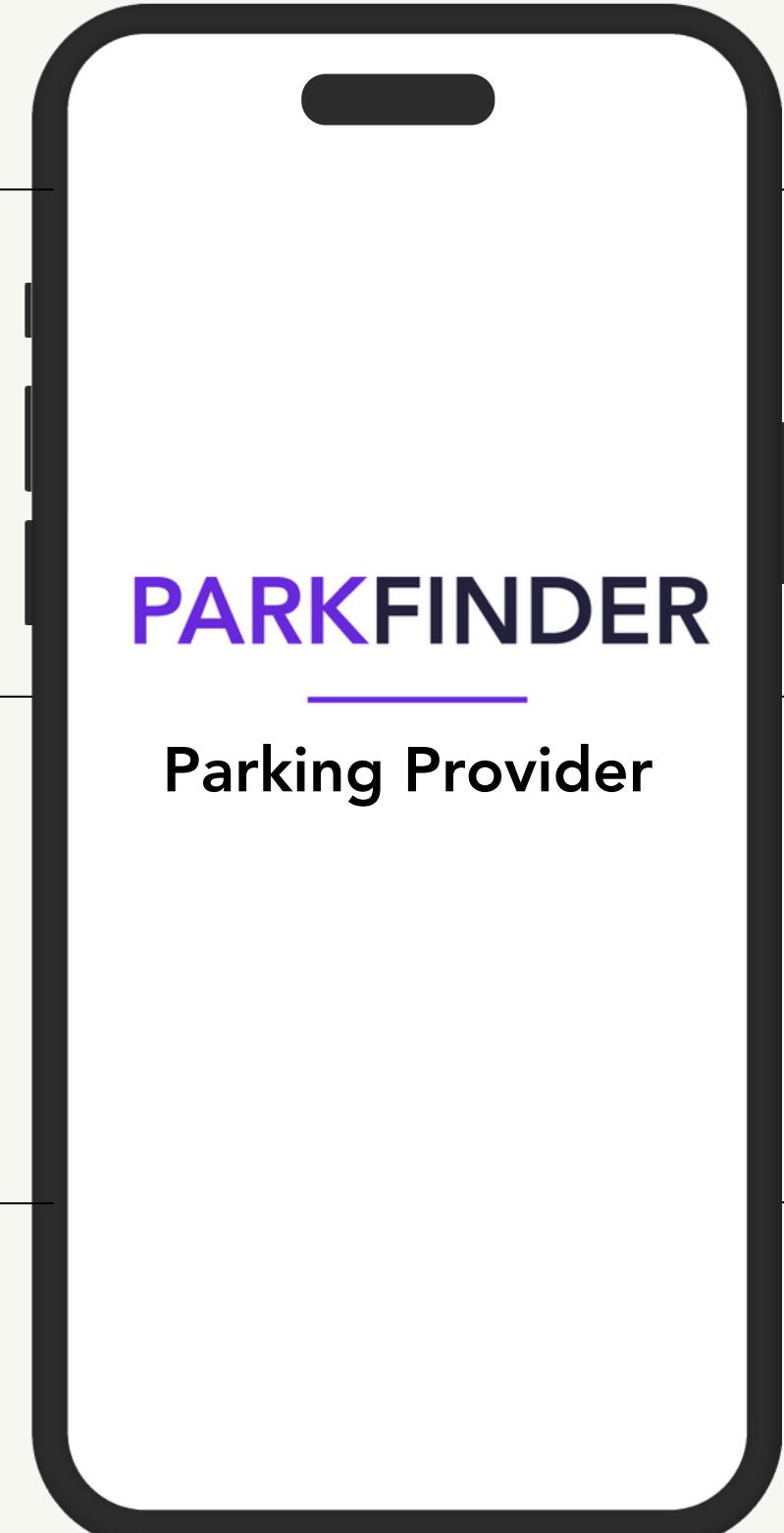
## สรุปรายได้

สามารถดูประวัติการให้เช่าและสรุปรายได้ของที่จอดรถได้



## ยืนยันการจองล่วงหน้า

สามารถยืนยันการจองล่วงหน้าได้ภายใน 1 วัน



## ส่งข้อความหาเจ้ารถ

สามารถส่งข้อความหาเจ้าของรถที่จอดอยู่ได้

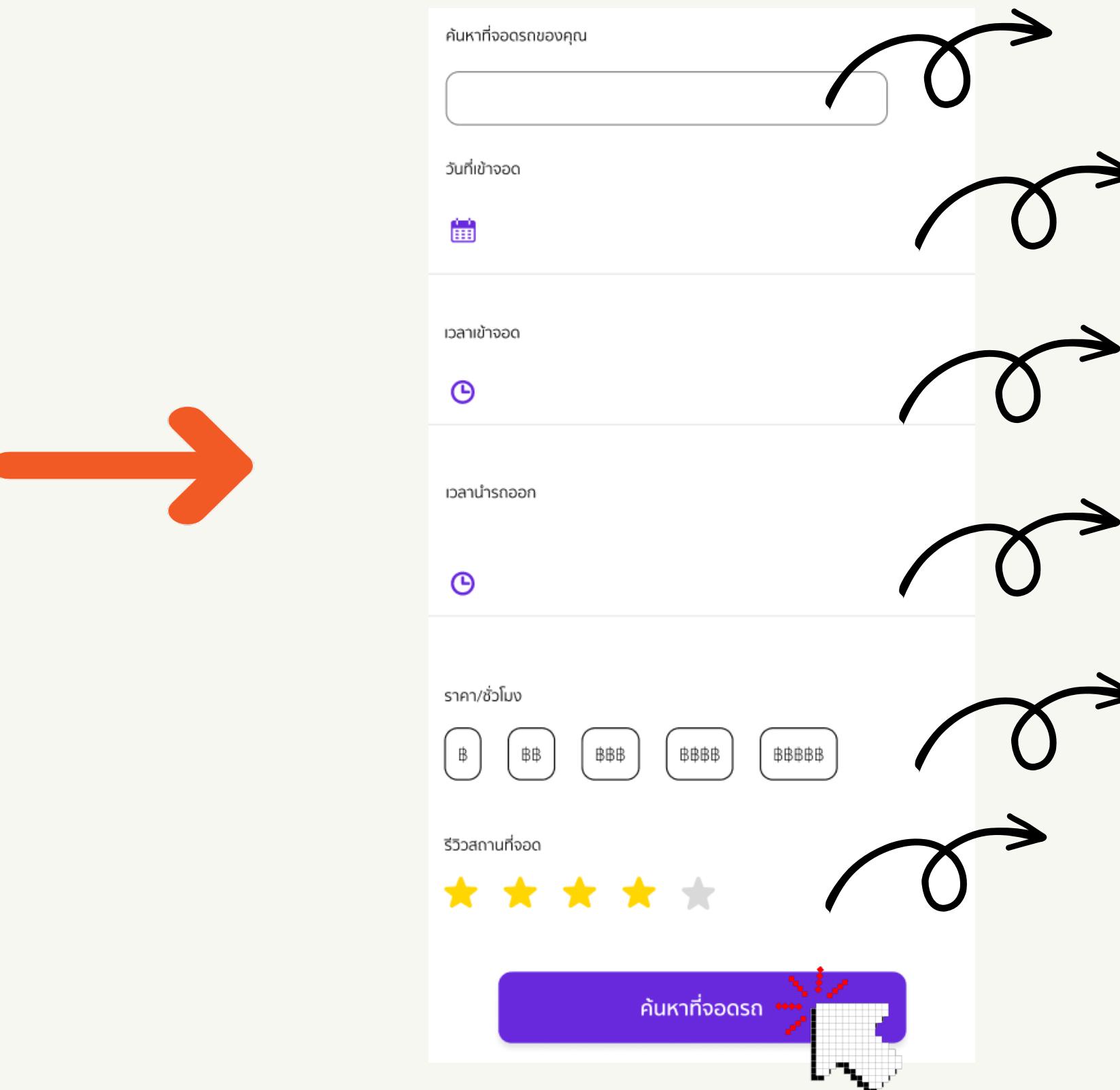
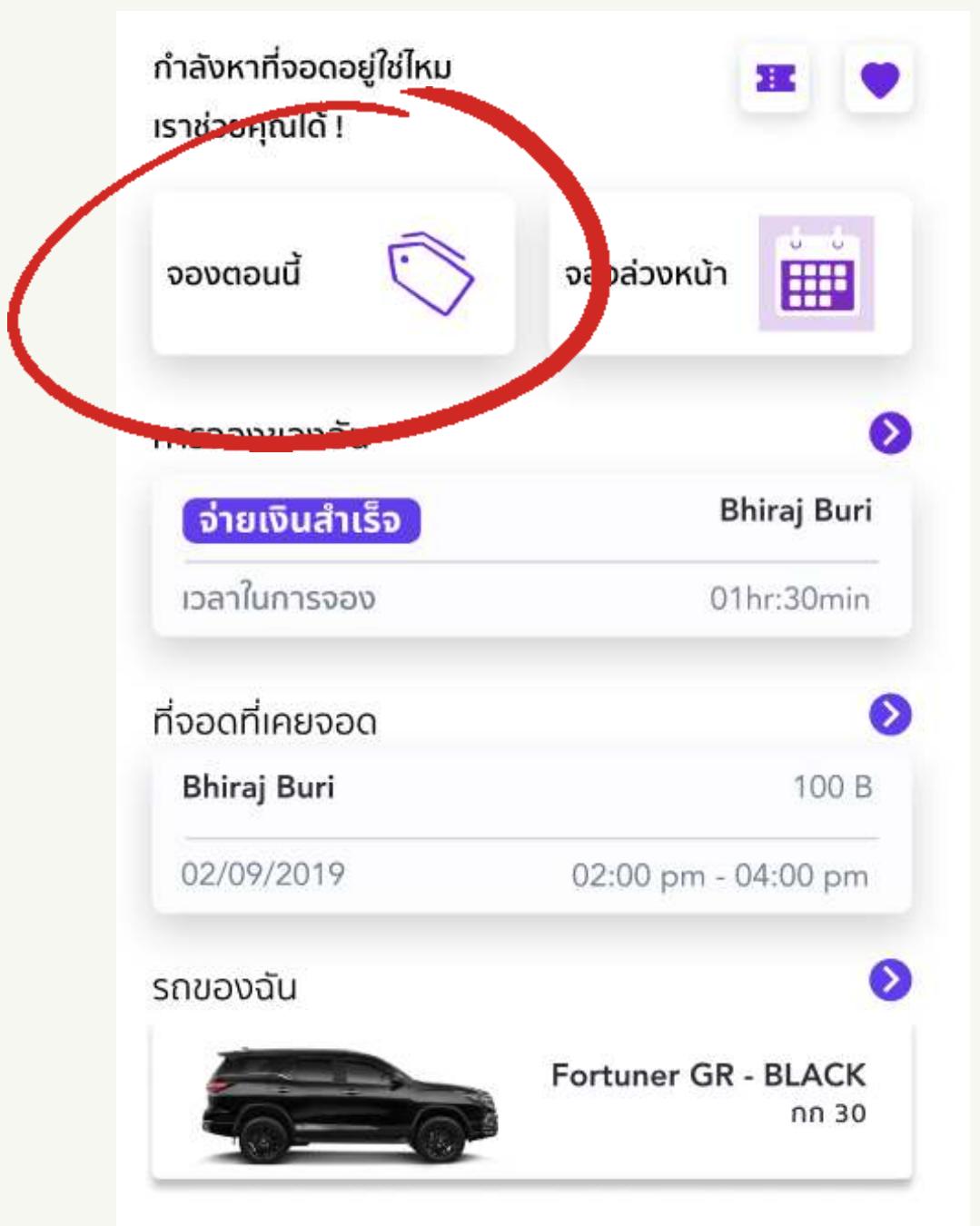


## แลก Rewards

สามารถดู และแลก Rewards ได้



# การจองที่จอดรถ



# ค้นหาจากสถานที่ชื่อดัง เช่น สยาม, หัวยขวาง

วันที่ต้องการเข้าจอด

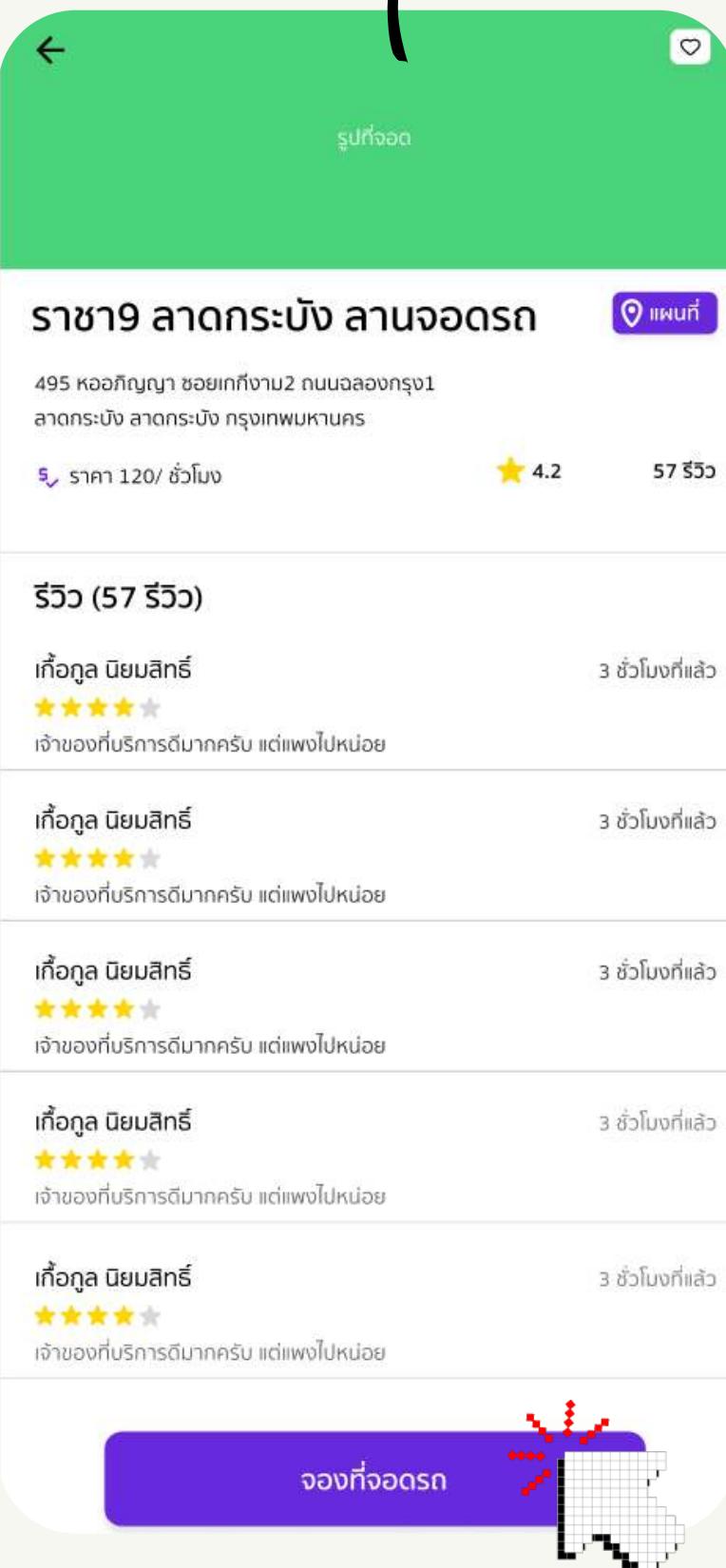
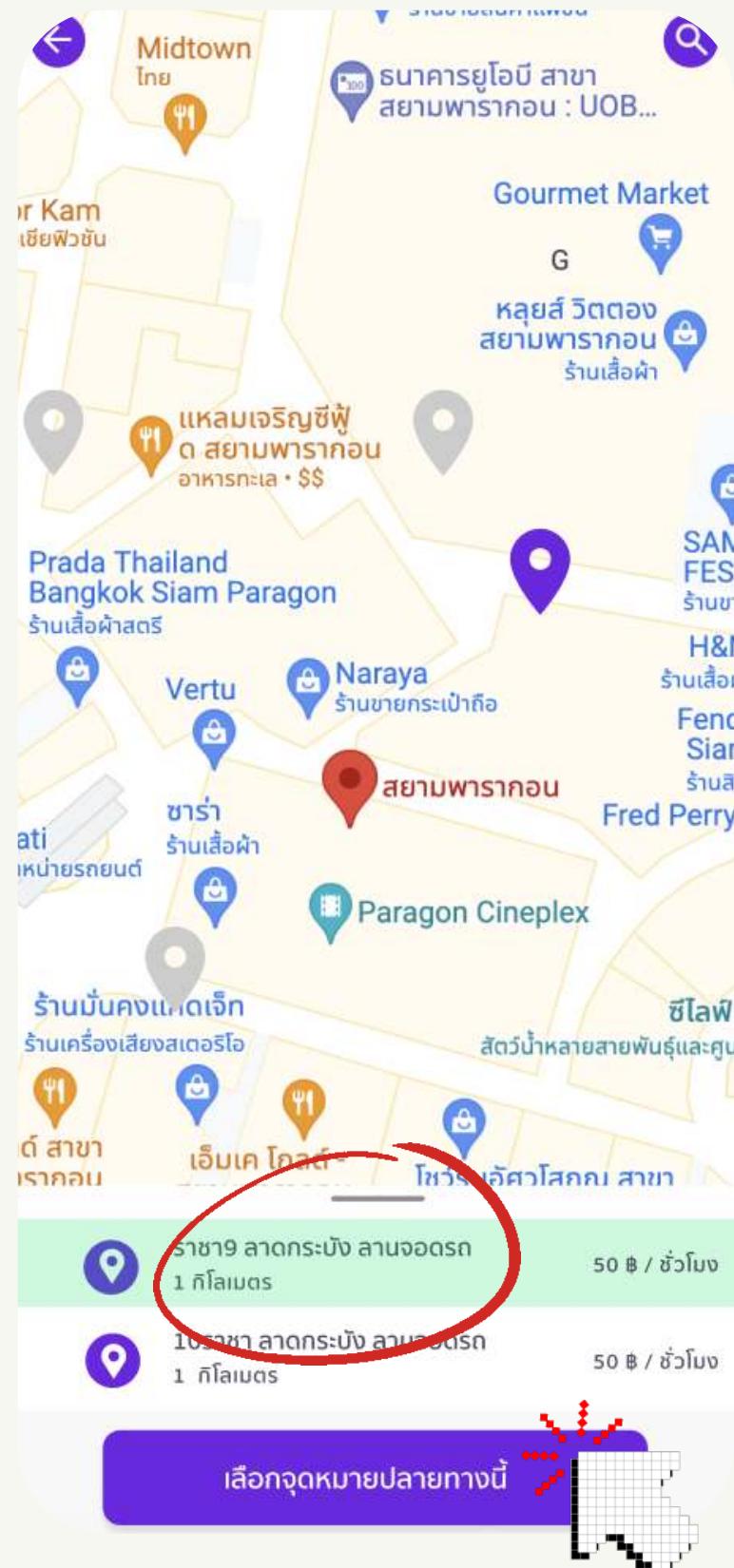
ເວລາທີ່ເຂົ້າຈອດ

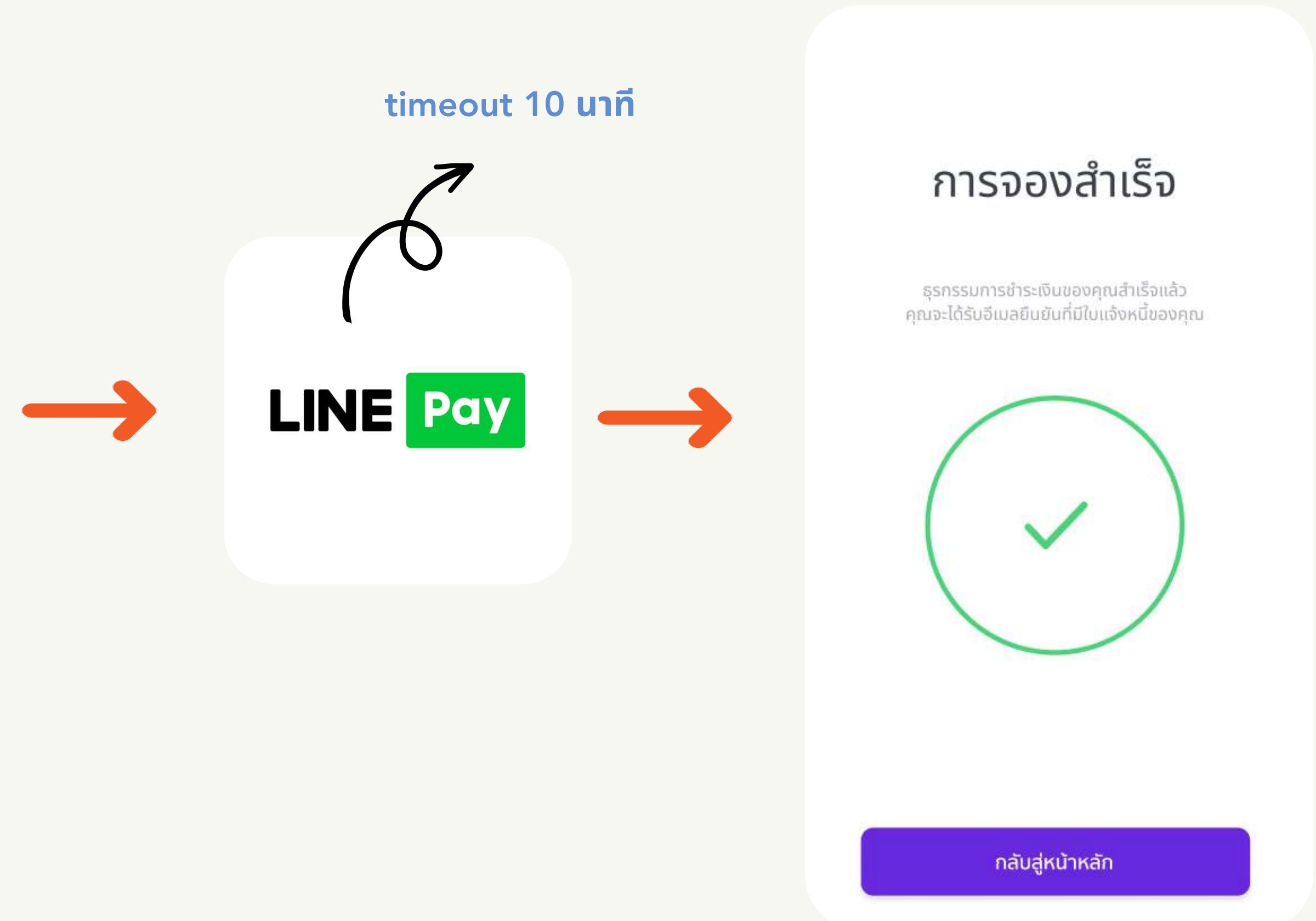
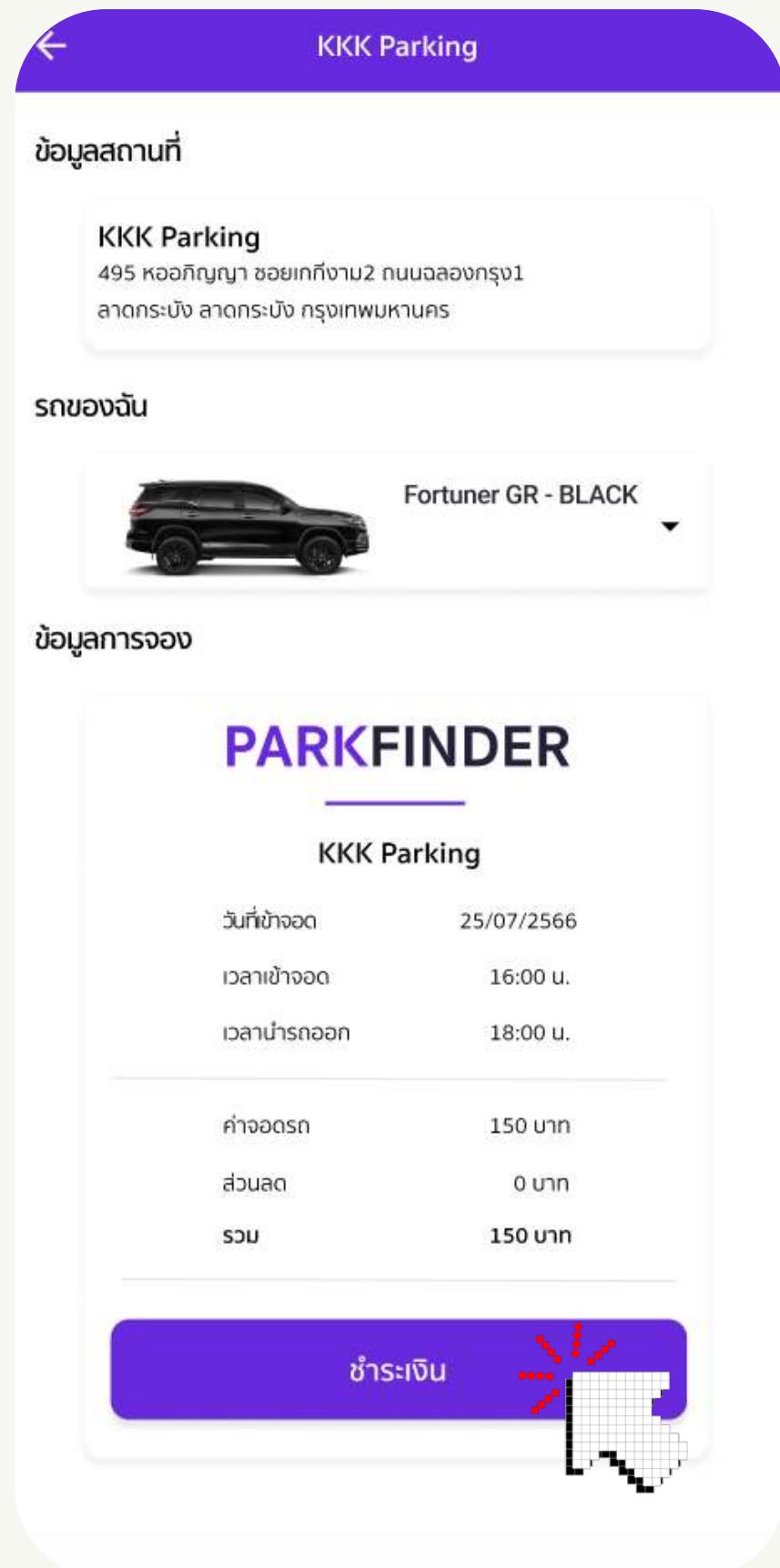
ເວລາທີຈະອອກ

ລາຍການ

ເຮັດວຽກ

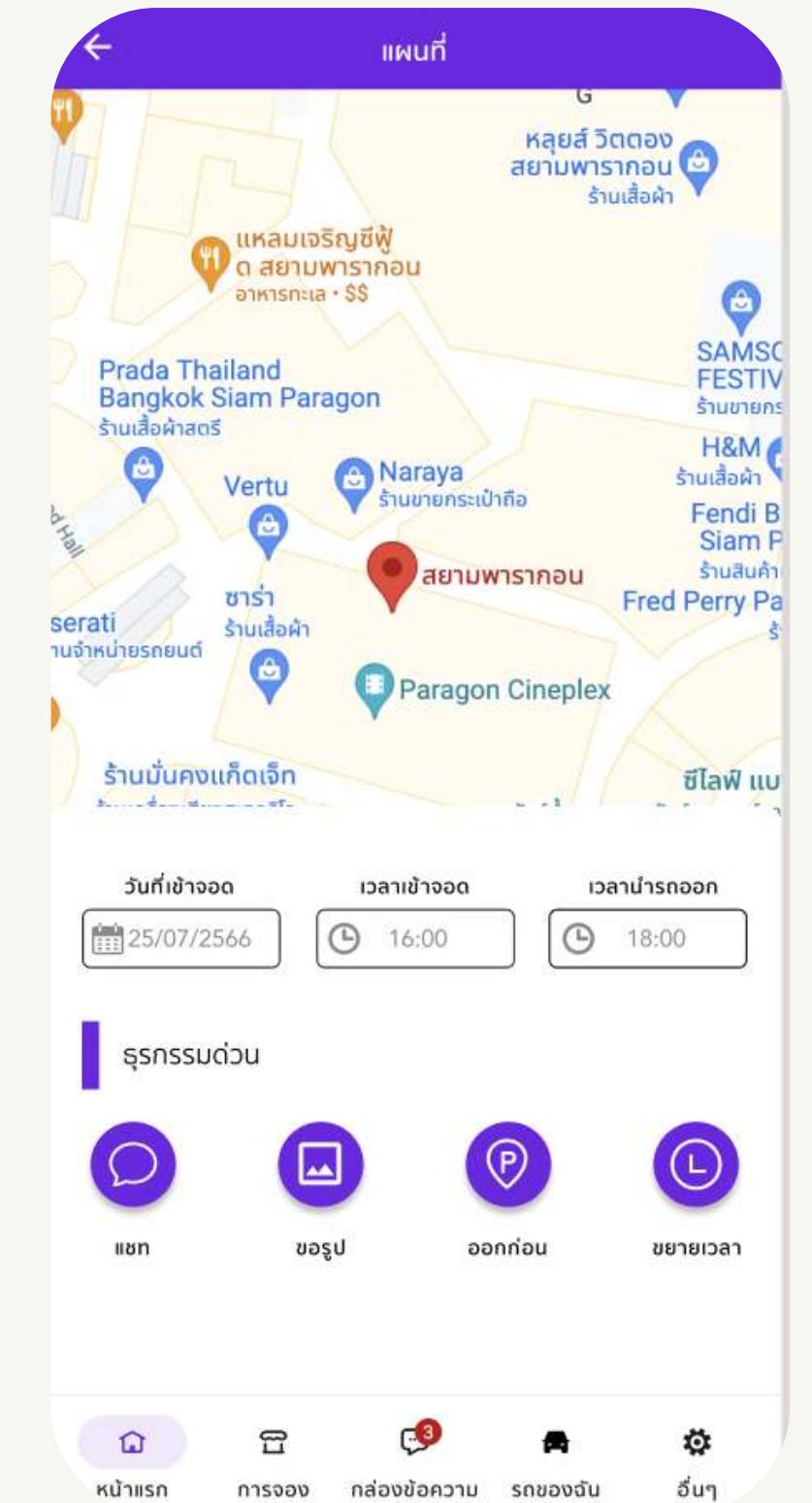
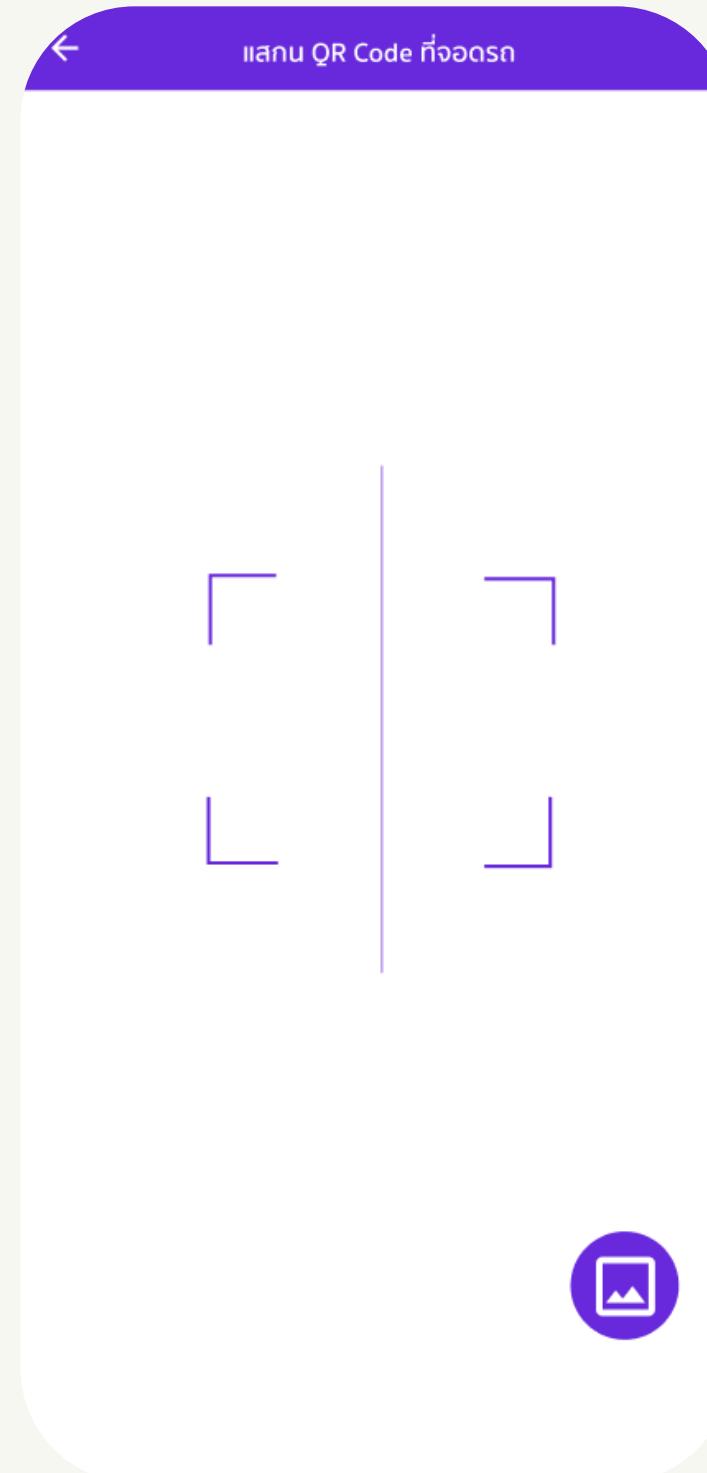
## ตรวจสอบความถูกต้อง







# การเข้าที่จอดรถ ของผู้ใช้งาน



## ● ออกรอบเวลา

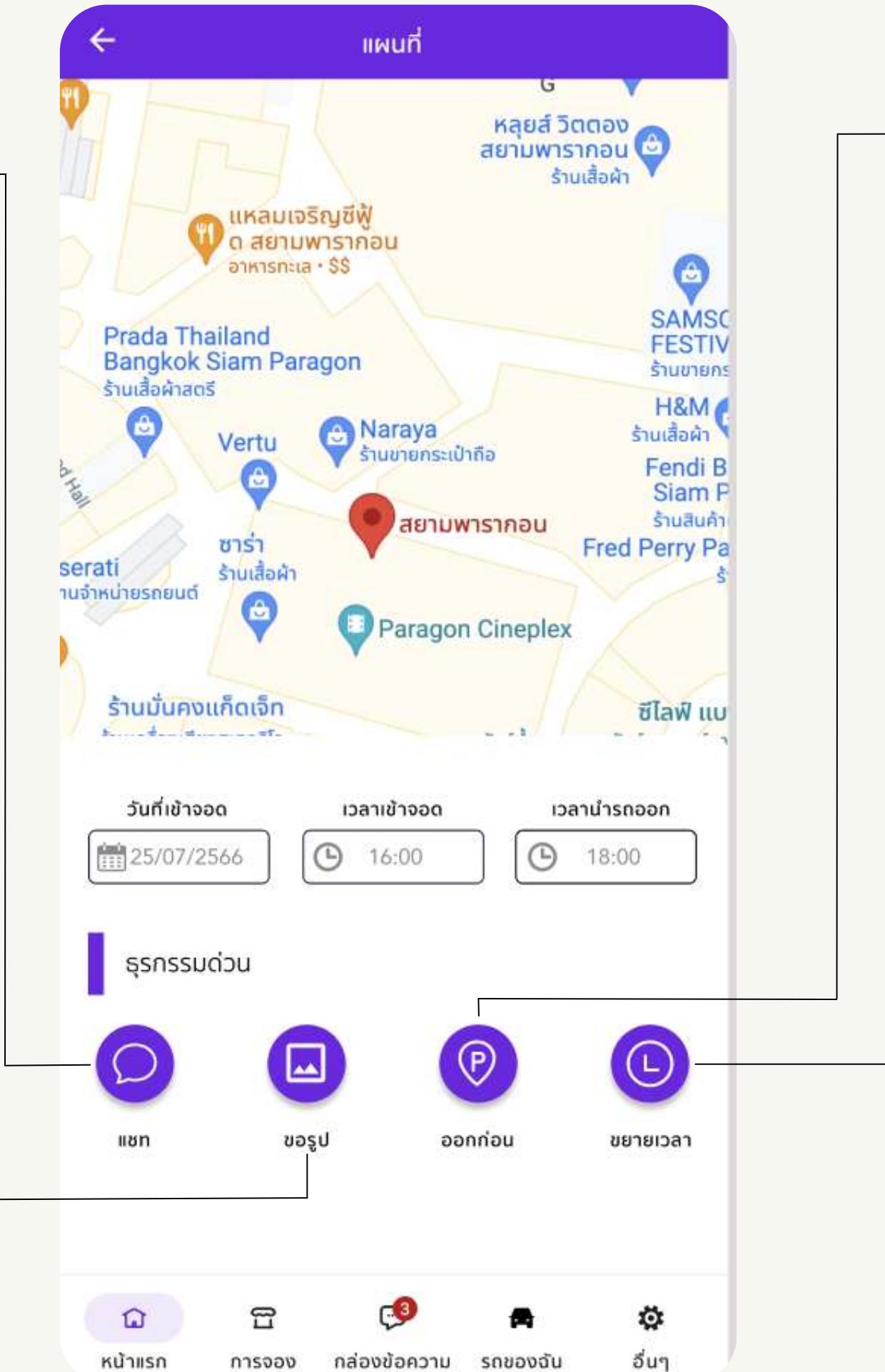
สามารถขออกรอบเวลาที่จะออกตามปกติ

### แบกกับเจ้าของที่จอดรถ ●

สามารถคุย สบทนา กับเจ้าของที่จอดรถได้  
อย่าง real-time

### สามารถขอรูปถ่ายที่จอดอยู่ ●

สามารถขอรูปถ่ายของตัวเองที่จอดอยู่  
ได้ตลอดเวลา



## ● ขยายเวลา

หากมั่นใจว่าไม่สามารถออกตามเวลาที่กำหนดได้ จะสามารถขยายเวลาโดยไม่มีค่าปรับ (กรณีที่มีคนจองต่อจะไม่สามารถขยายเวลาได้)



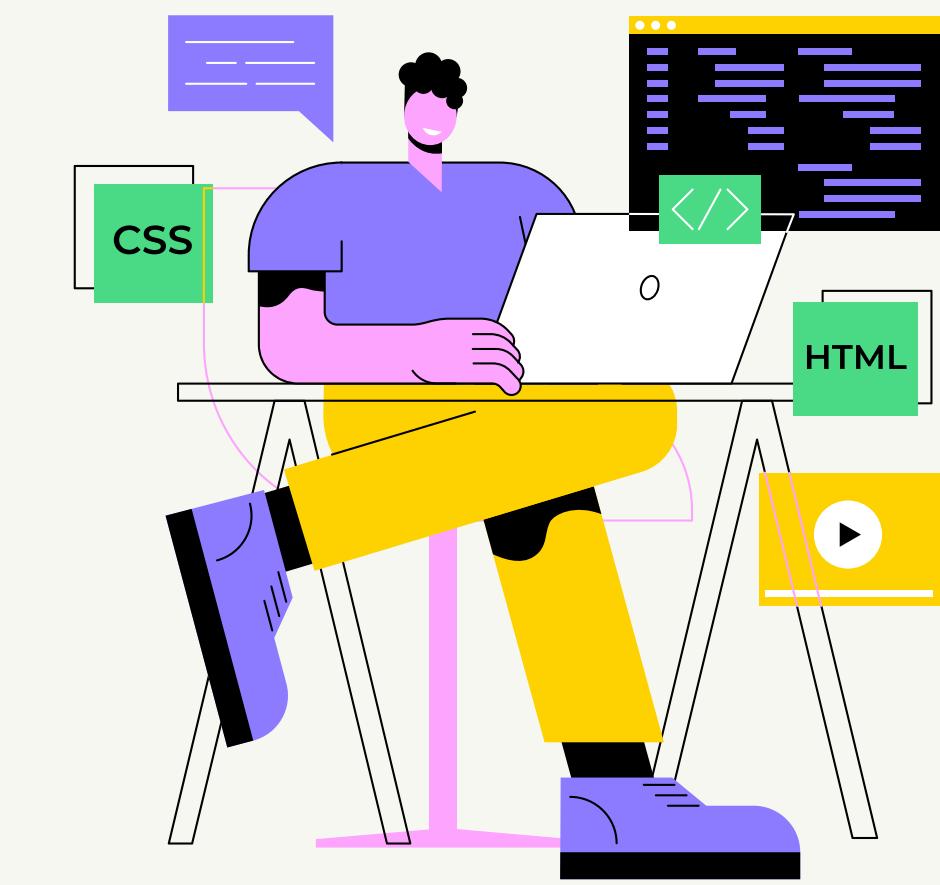
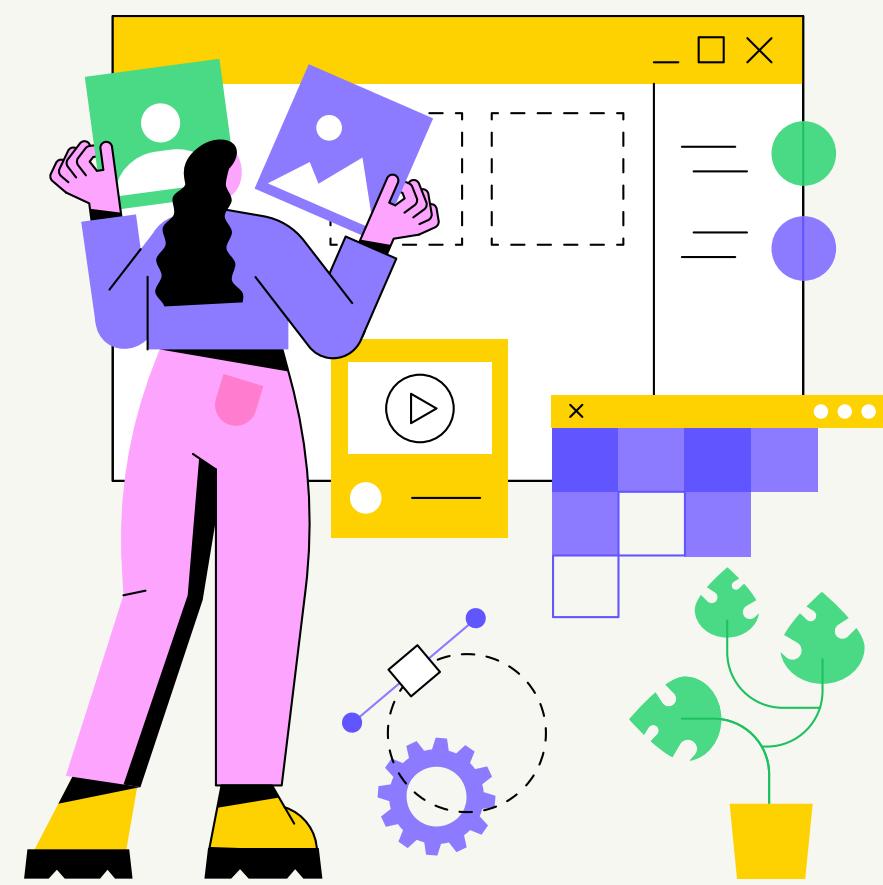
# Progress

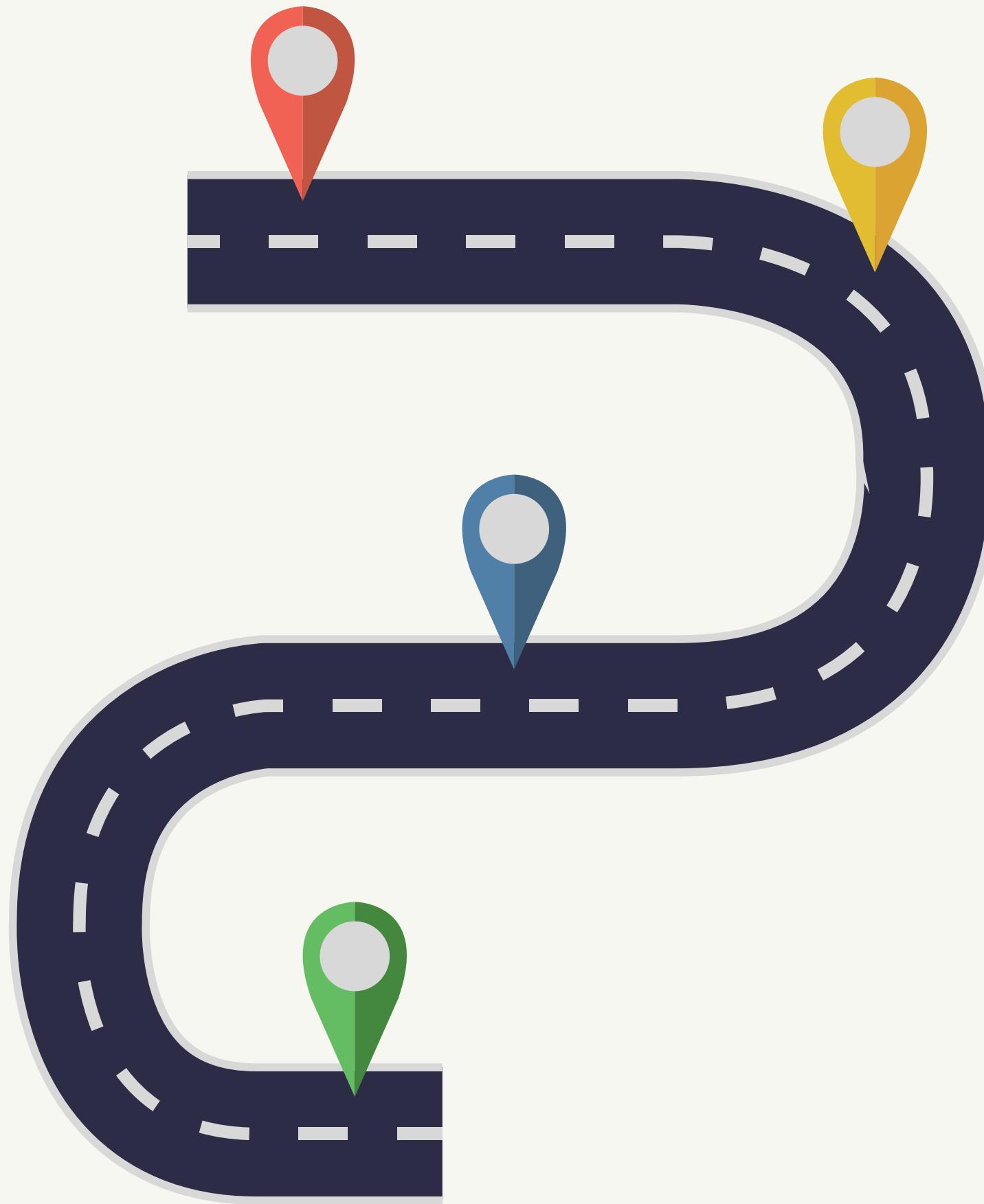


**FRONT END**  
60 %

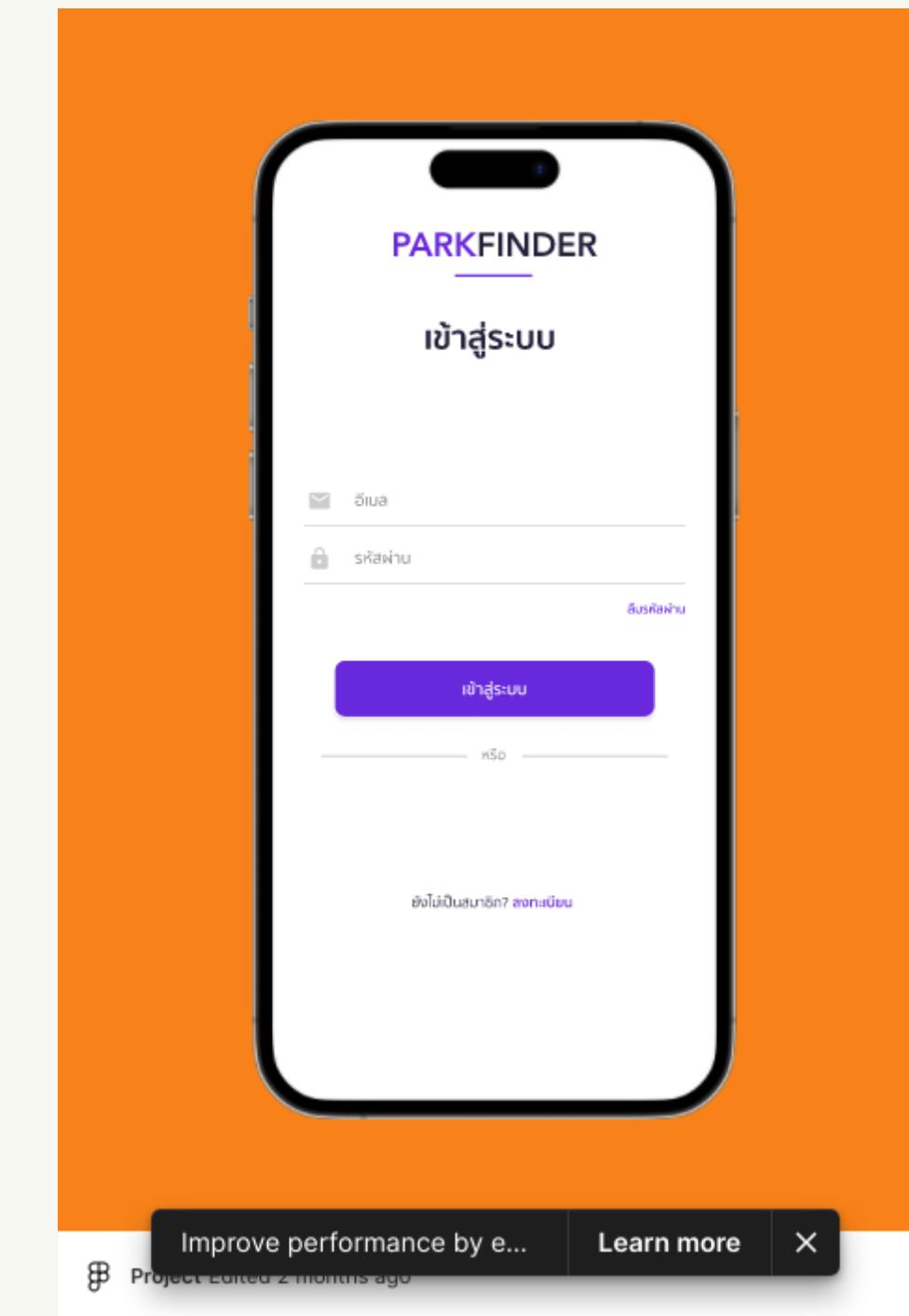
**UXUI**

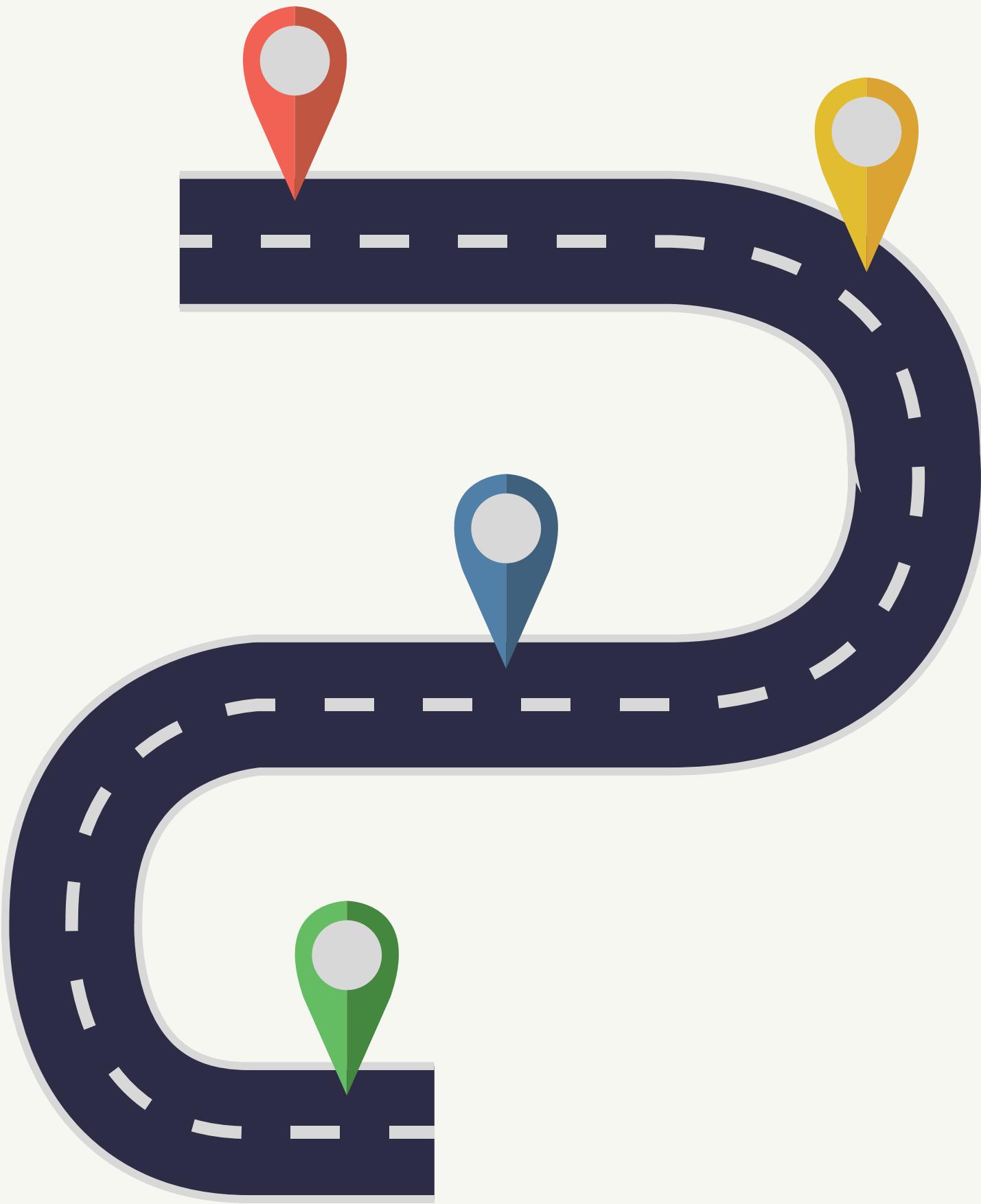
40 %





The logo consists of a red location pin icon on the left, followed by a vertical grey bar, and then the text 'UXUI' in large blue block letters above the word 'design' in a larger blue sans-serif font.





# Login Register

The image displays three sequential screens from a mobile application named "PARKFINDER".

**Screen 1: Login/Register Page**

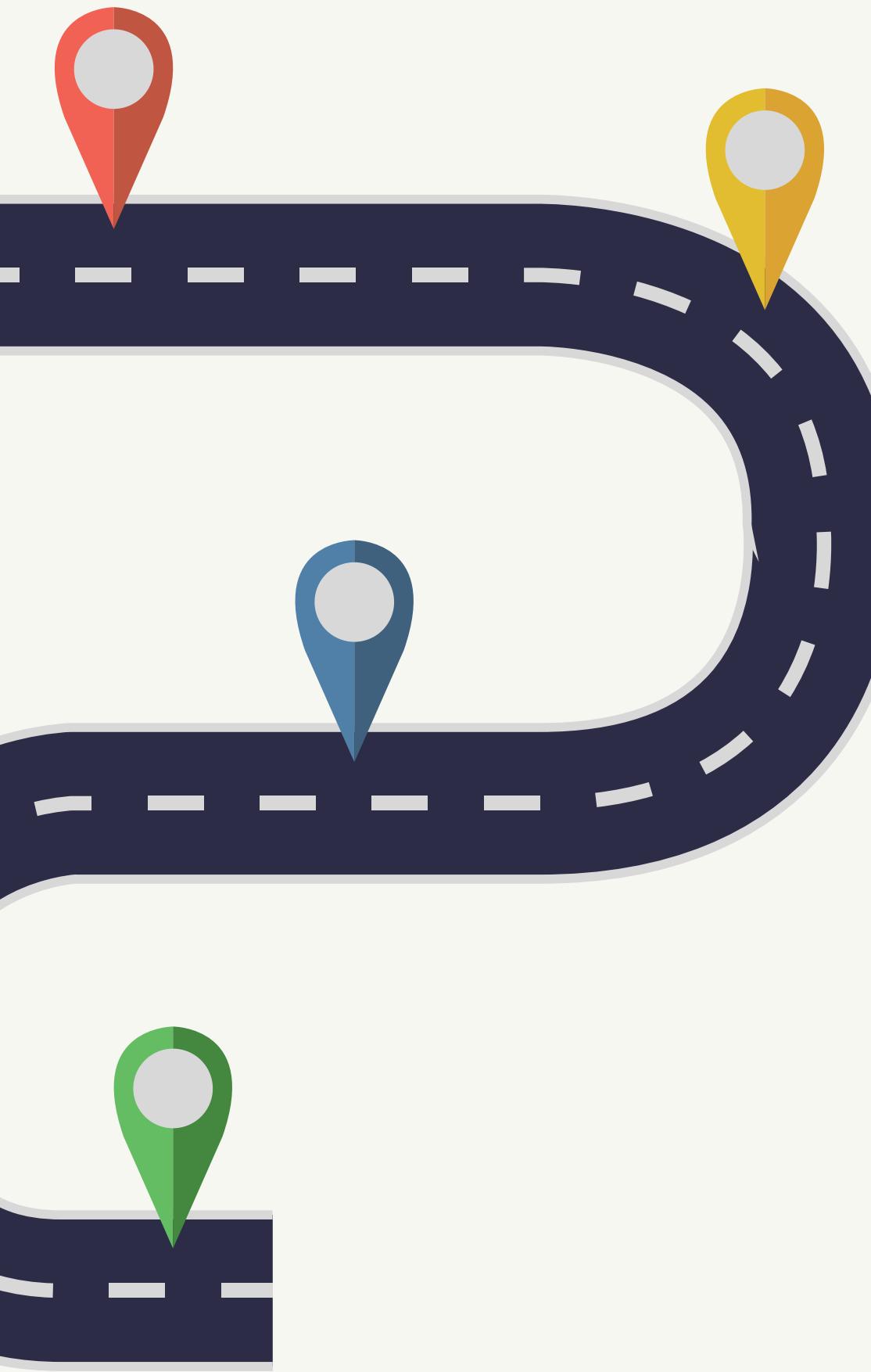
The title "PARKFINDER" is at the top. Below it, there are two large buttons: "เข้าสู่ระบบ" (Log In) in purple and "ลงทะเบียน" (Sign Up) in green. To the left of these buttons are input fields for "อีเมล" (Email) and "รหัสผ่าน" (Password). Below the input fields is a "ลืมรหัสผ่าน" (Forgot Password) link. At the bottom are "เข้าสู่ระบบ" and "ลงทะเบียน" buttons again, with a "เข้าสู่ระบบด้วย Line" (Log In with Line) button in between.

**Screen 2: Registration Step 1**

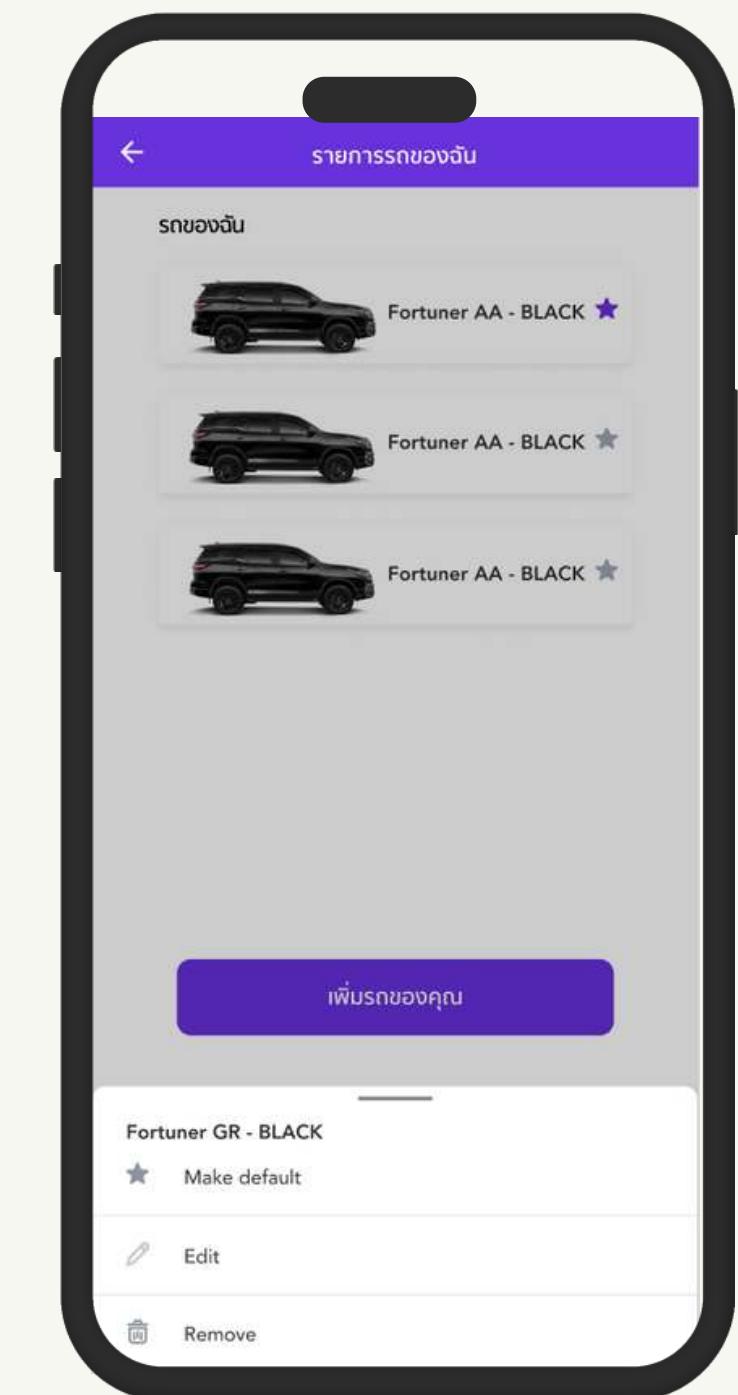
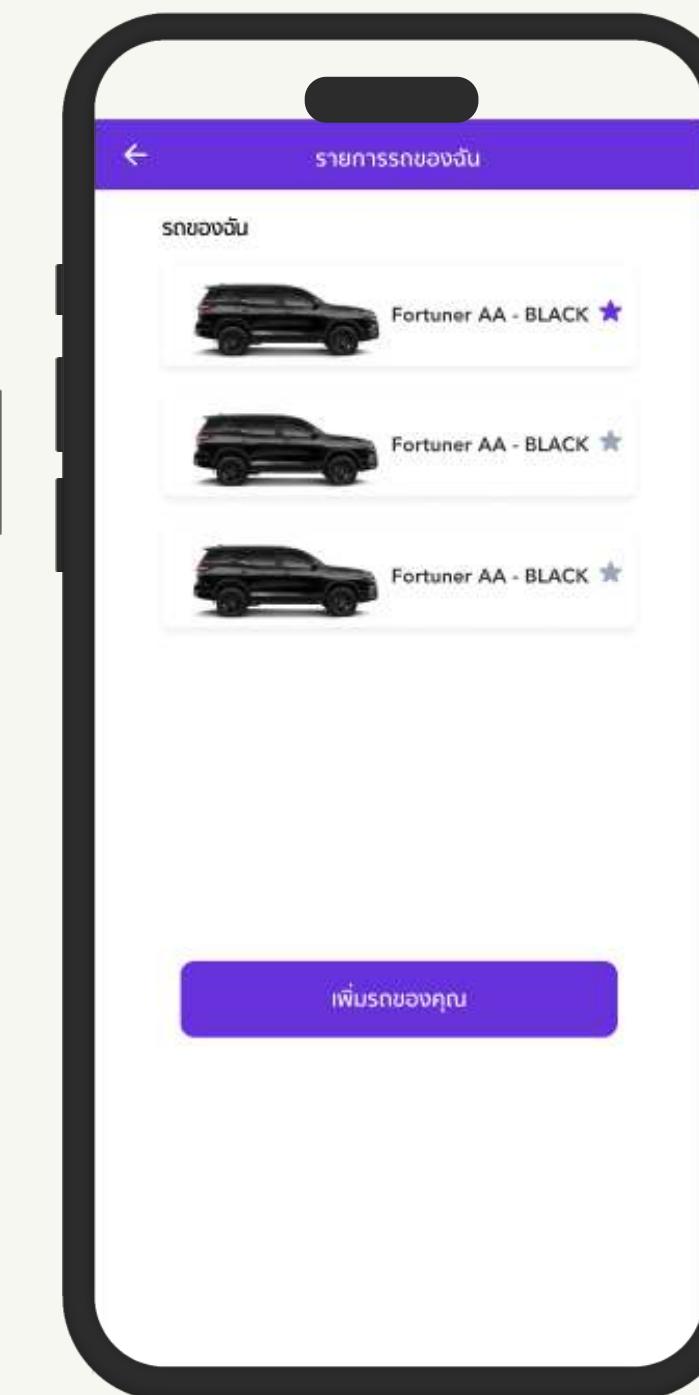
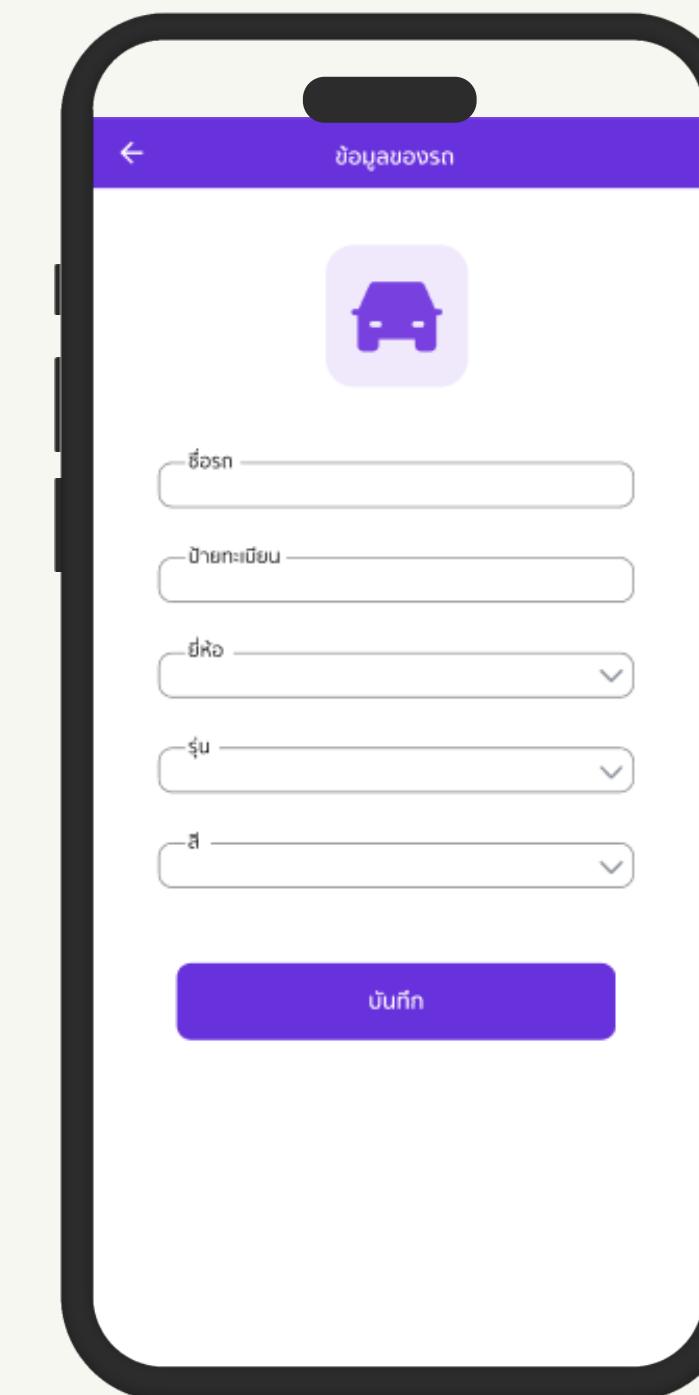
The title "PARKFINDER" is at the top. Below it, there are six input fields: "ชื่อ" (Name), "นามสกุล" (Surname), "เบอร์โทรศัพท์" (Phone Number), "อีเมล" (Email), "รหัสผ่าน" (Password), and "ยืนยันรหัสผ่าน" (Confirm Password). Below these fields are "ลงทะเบียน" and "เป็นสมาชิกอยู่แล้ว? เข้าสู่ระบบ" (Already a member? Log In) buttons.

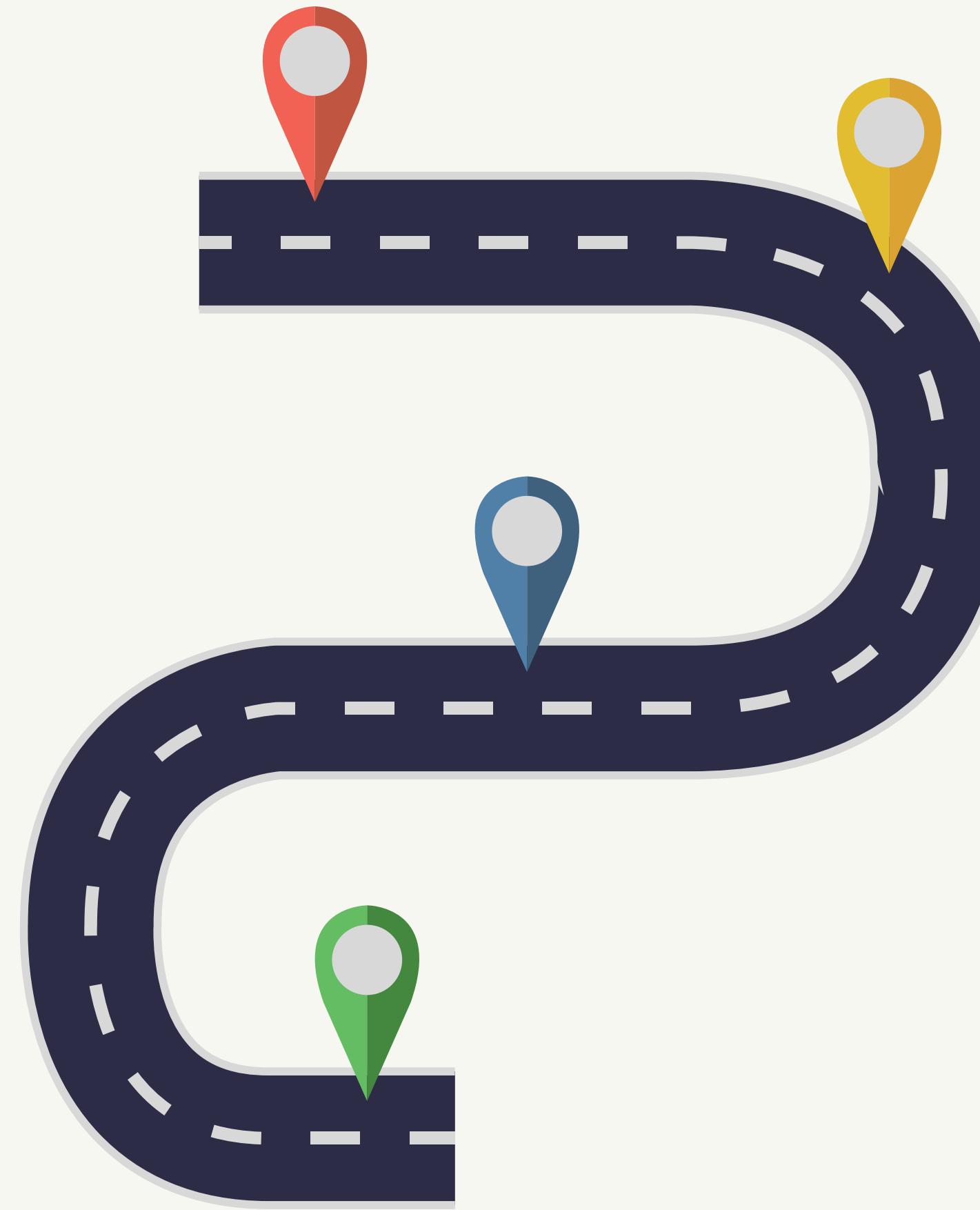
**Screen 3: Registration Step 2**

The title "การยืนยันขั้นที่ 2" (Step 2 Confirmation) is at the top. It shows a purple padlock icon and the text "กรุณากรอกรหัส OTP ที่ส่งไปในอีเมล" (Please enter the OTP sent to your email). Below this are buttons for "ยืนยัน" (Confirm) and "ไม่ได้รับรหัส OTP? ส่งอีกครั้ง" (Didn't receive OTP? Send again).

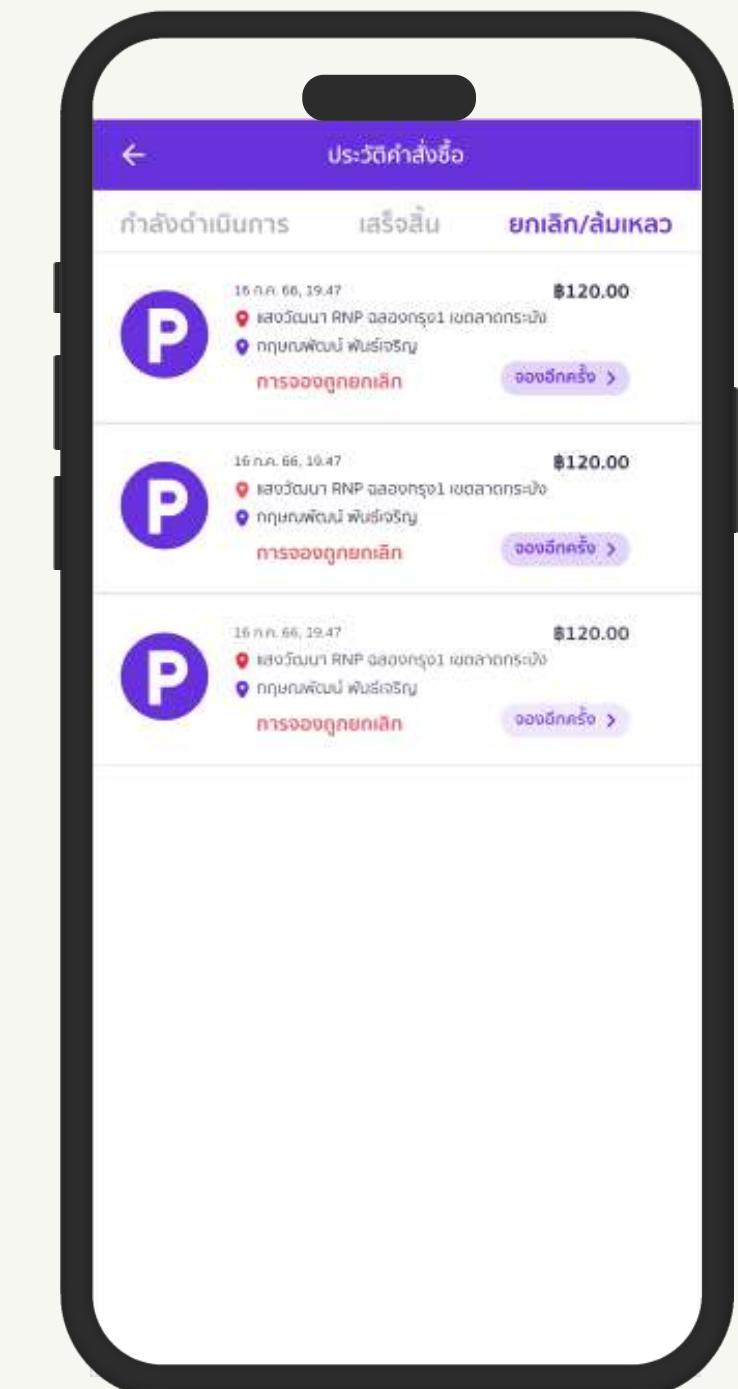
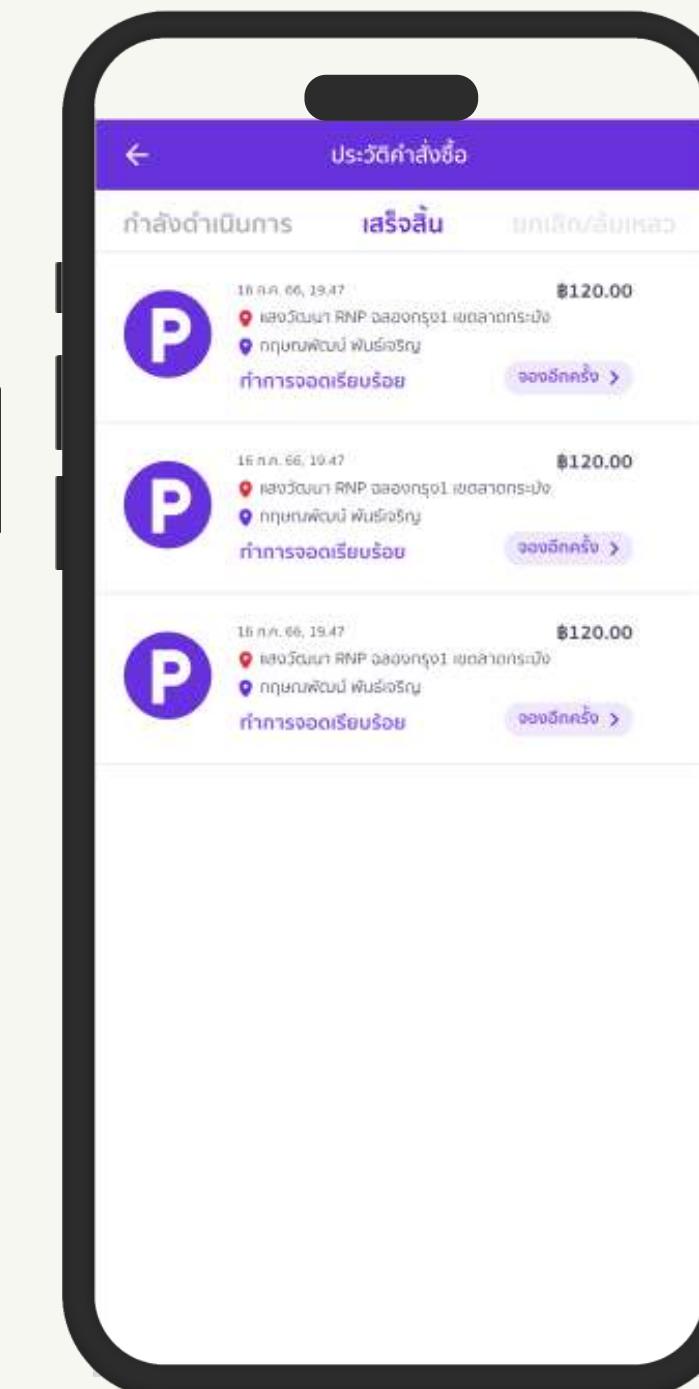
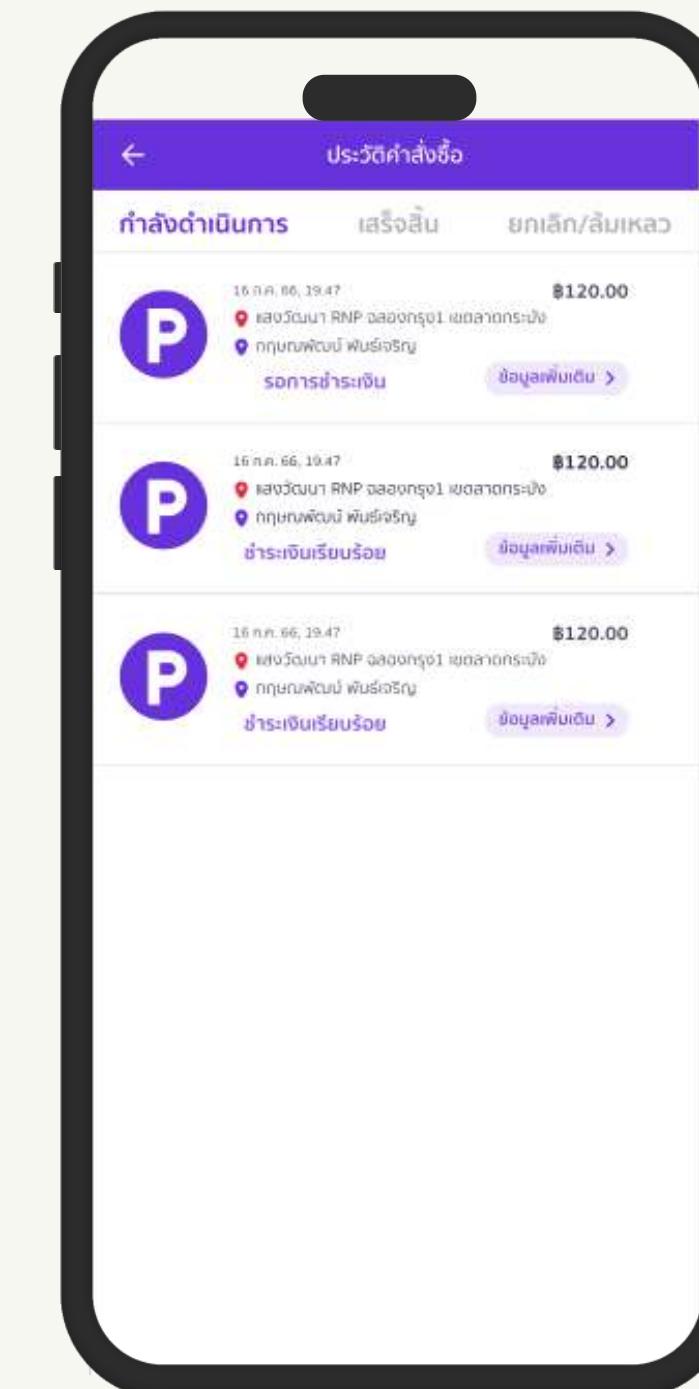


# My Car





# History

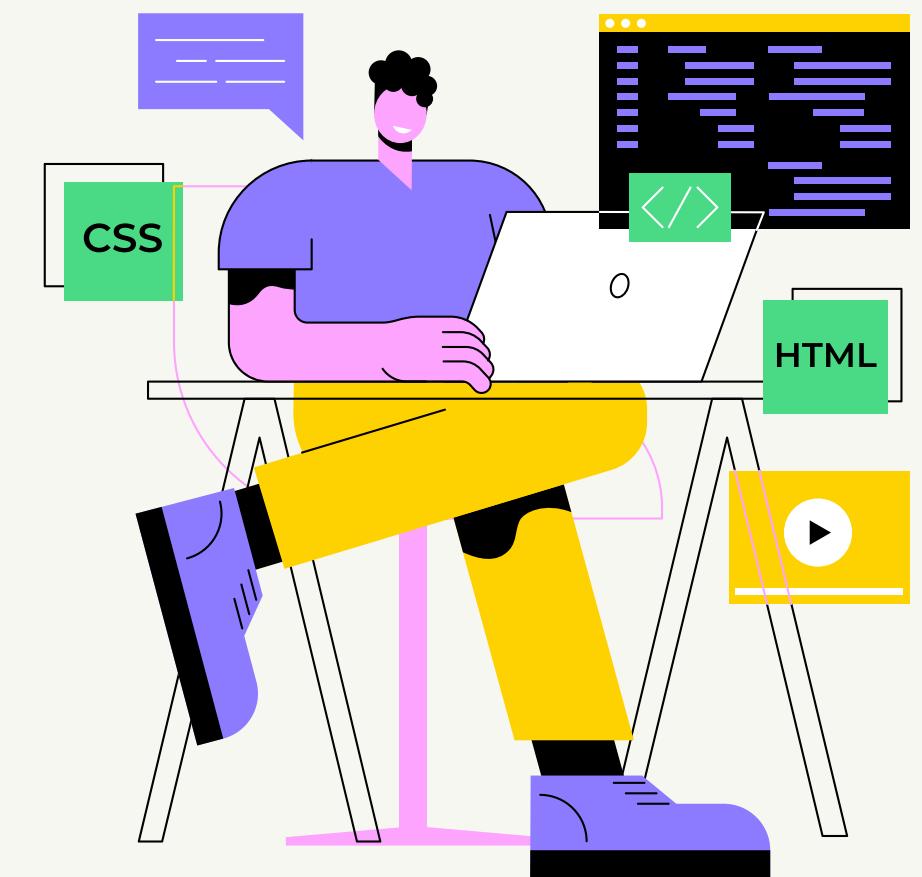


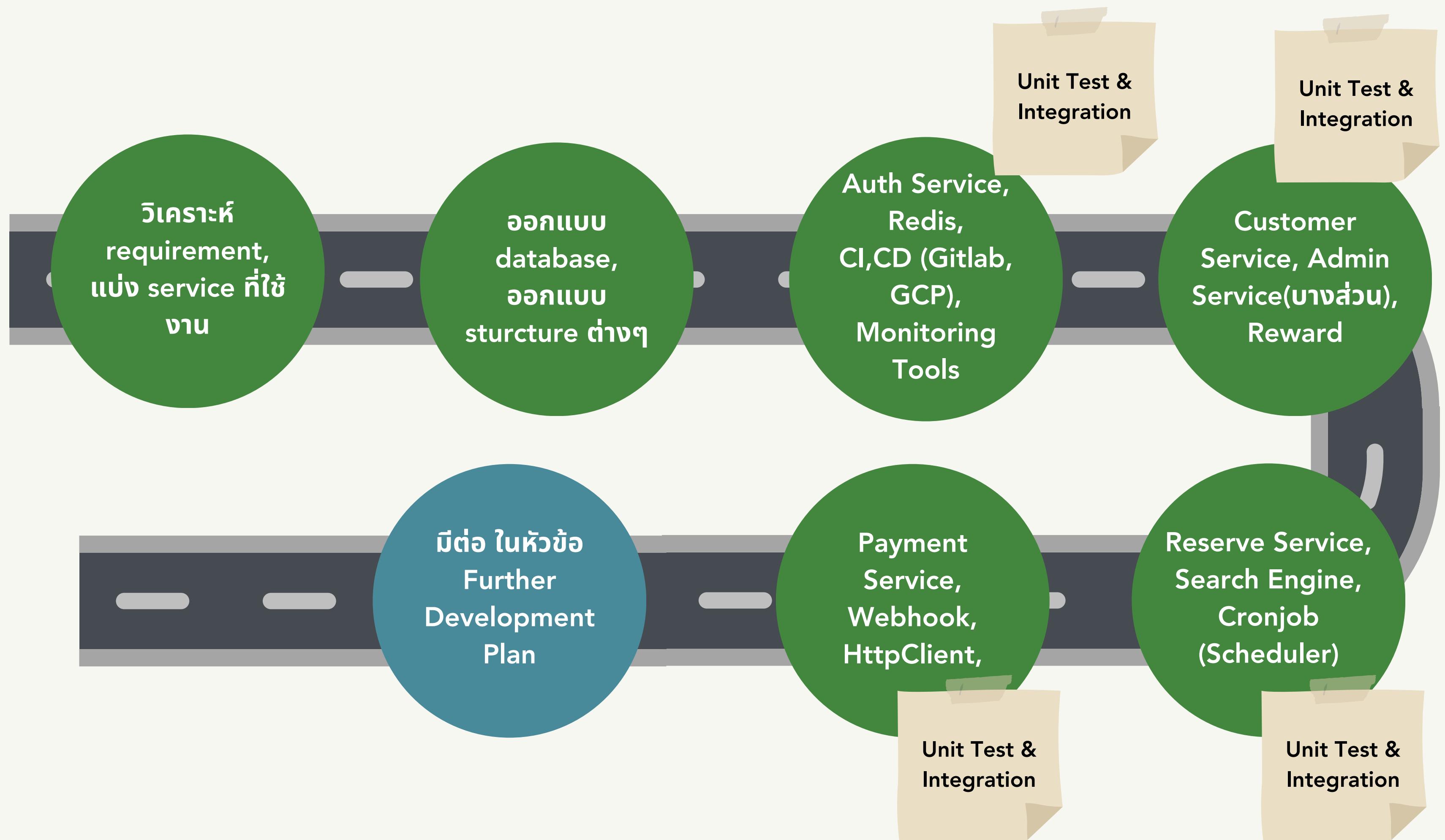
# DEPLOYMENT

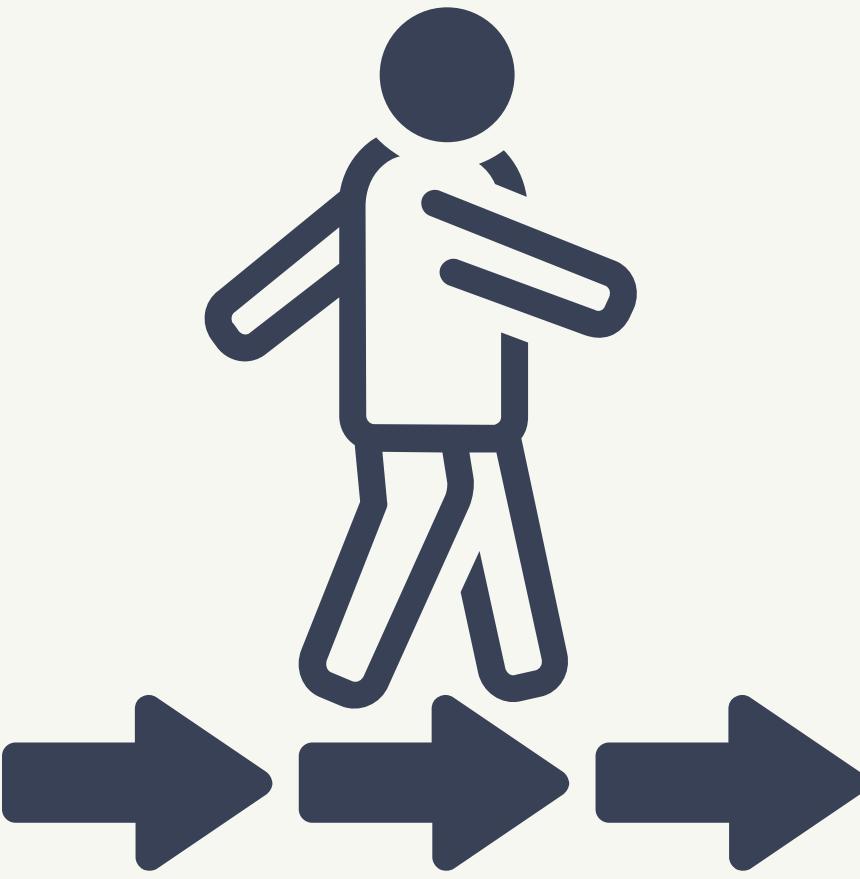
30 %



BACK END  
70 %







# Further Development Plan



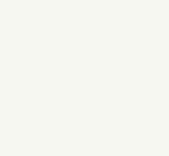
## UXUI

- Parking Provider
- Admin Web App



## Integration

- Provider App
- Customer App
- Admin Web App

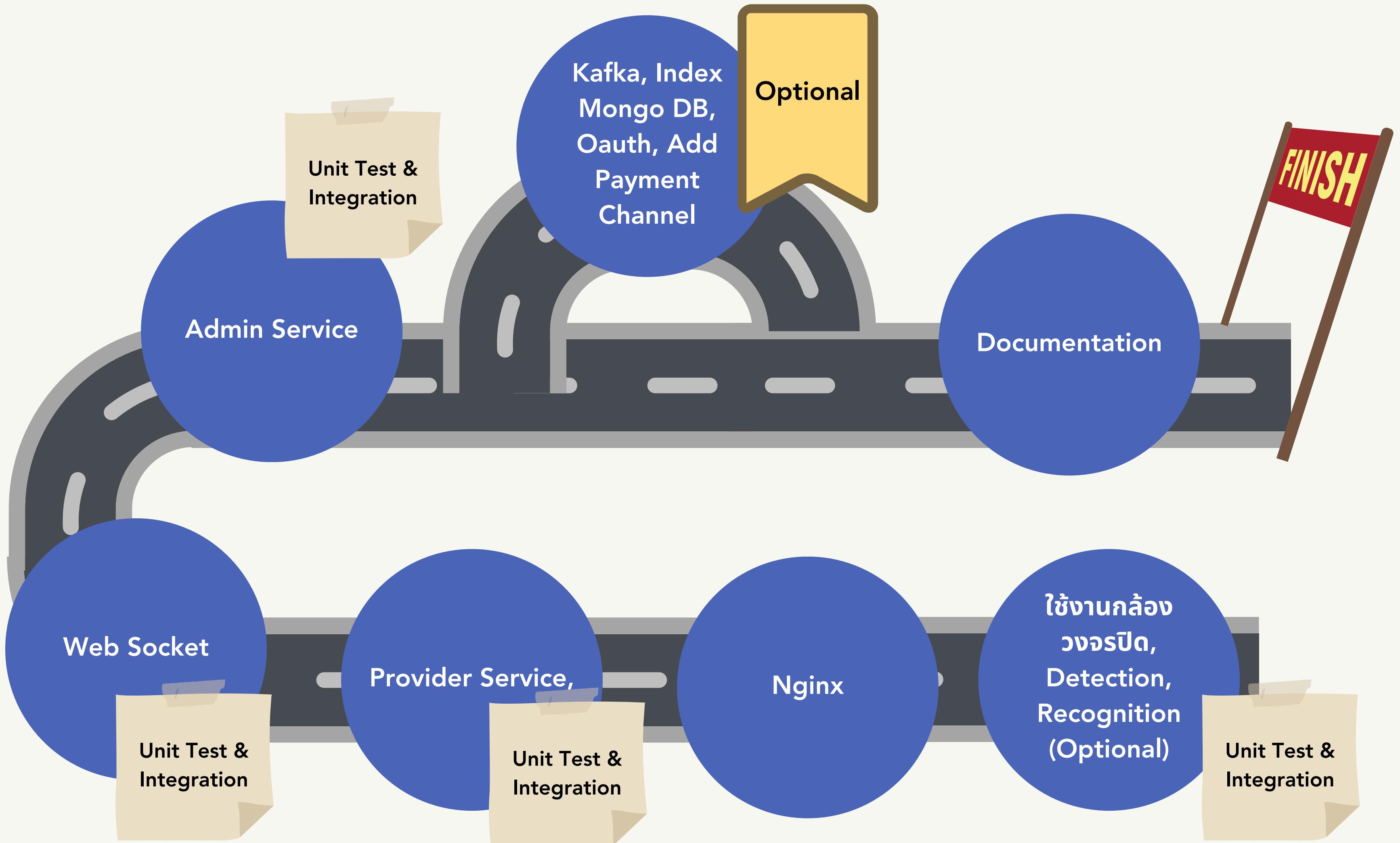


## Front end

- Provider App
- Admin Web App



## Automated test



# Thanks for your attention!

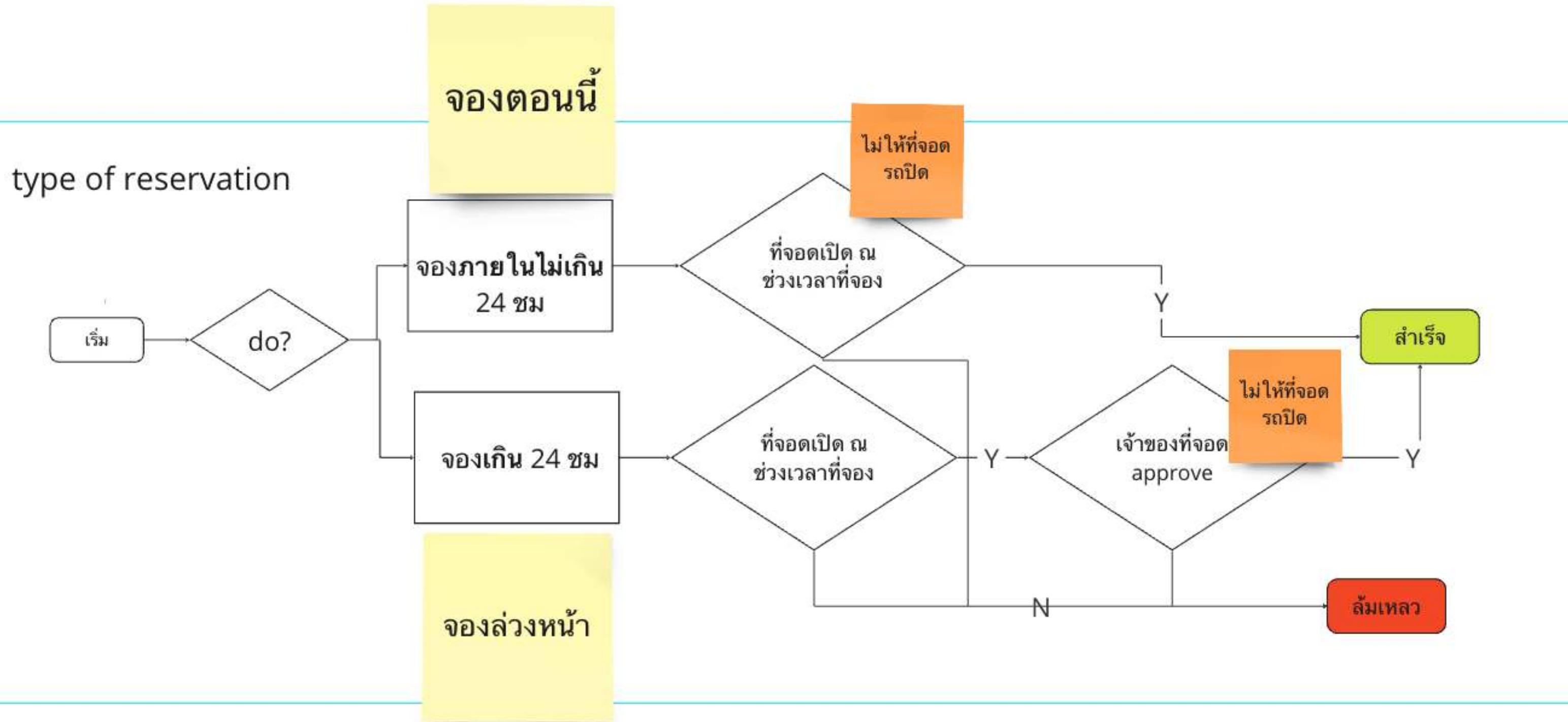


# Appendix



# GENERAL

# จองต่อนี่ กับ จองล่วงหน้าต่างกันอย่างไร



# จะแก้ปัญหากรณีคนจอดรถคันเก่าออกไม่ตามเวลา และมีคนจองต่ออยังไง?

1

มีการป้องกันไว้เบื้องต้นคือ ในการจองจะไม่สามารถ  
จองติดกันได้เป็นเวลา 1 ชั่วโมง  
เช่น A จอง 12.00-13.00 B จะเริ่มได้ตั้งแต่ 14.00  
หรือ จองได้ถึงแค่ 11.00

2

ชั่งระบบได้เพื่อเวลาให้นาย A ในการเรทเป็นเวลา  
ทั้งสิ้น 15 นาทีโดยไม่เสียค่าปรับ

## จะแก้ปัญหากรณีคนจอดรถคันเก่าออกไม่ตามเวลา และมีคนจองต่ออยังไง?(ต่อ)

3

แต่หากนาย A รู้ว่าไม่สามารถออกก่อนเวลาได้ และ  
ขยายเวลาไม่ได้เนื่องจากนาย B จองไว้ต่อน 14.00  
นาย A ต้องยอมเสียค่าปรับเพื่อให้ระบบนำเงินไป  
ชดเชยให้นาย B

4

การที่เว้นเวลาไว้ 1 ชั่วโมงทำให้เราสามารถมีเวลาให้  
นาย B ทราบถึงปัญหาและมีเวลาหาที่จอดใหม่ถึง  
45 นาทีเลยทีเดียว

# CAMERA

## วิธีที่จะทำงานร่วมกับกล้องที่ research ไว้เบื้องต้น

ขั้นตอน: หา API ดึงรูปจากกล้อง

ในบทความนี้จะใช้ภาพนิ่งจากกล้อง Hikvision ซึ่งหลายๆ model ที่ขายในบ้านเรา จะใช้ Pattern ของ url ประมาณนี้:

<http://{Ip Address}/Streaming/channels/{CameraId}/picture>

Note: ยื่ห้ออื่นๆที่น่าจะเจอกันบ่อย  
dahua:

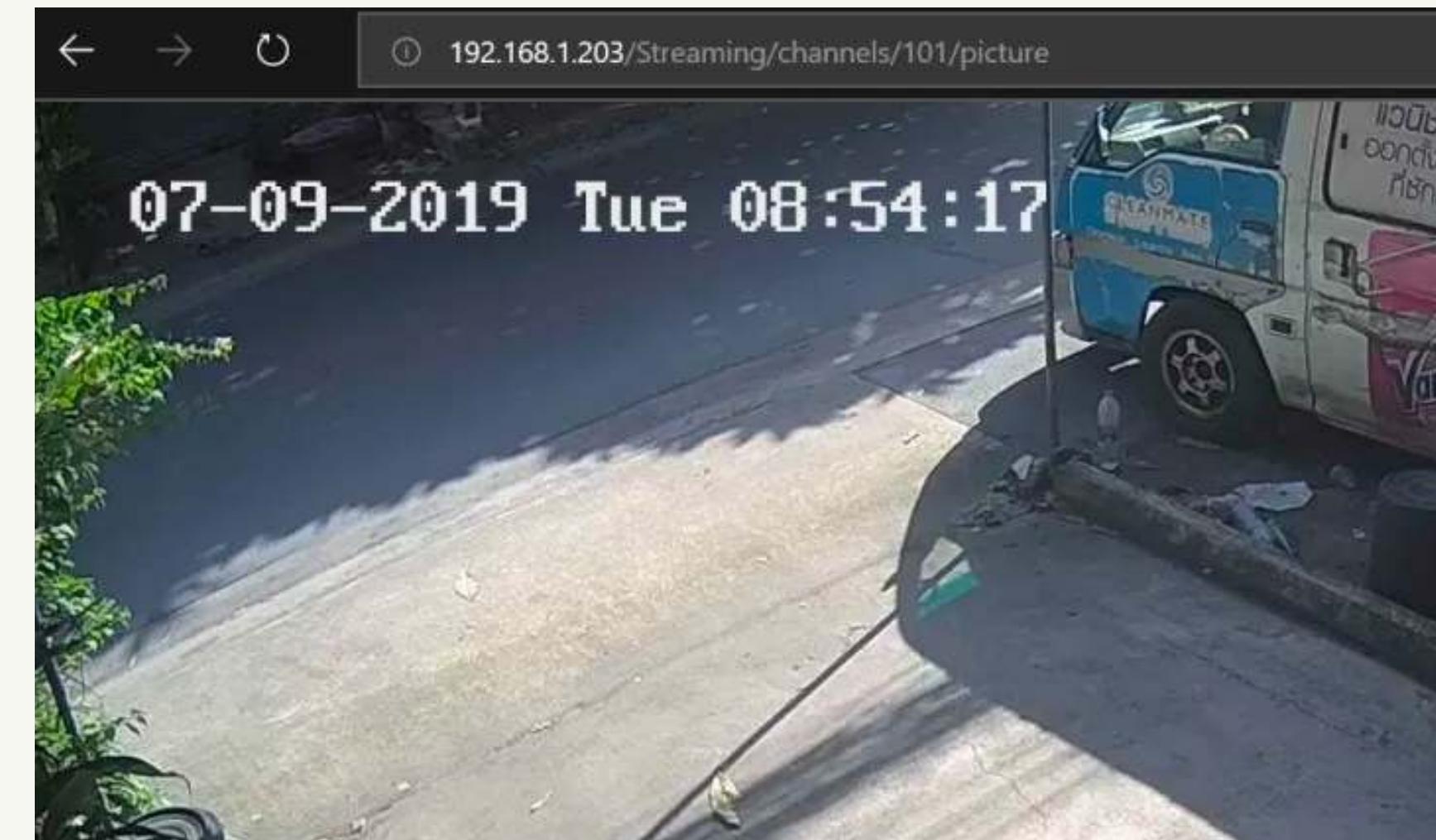
<http://{Ip Address}/cgi-bin/snapshot.cgi>

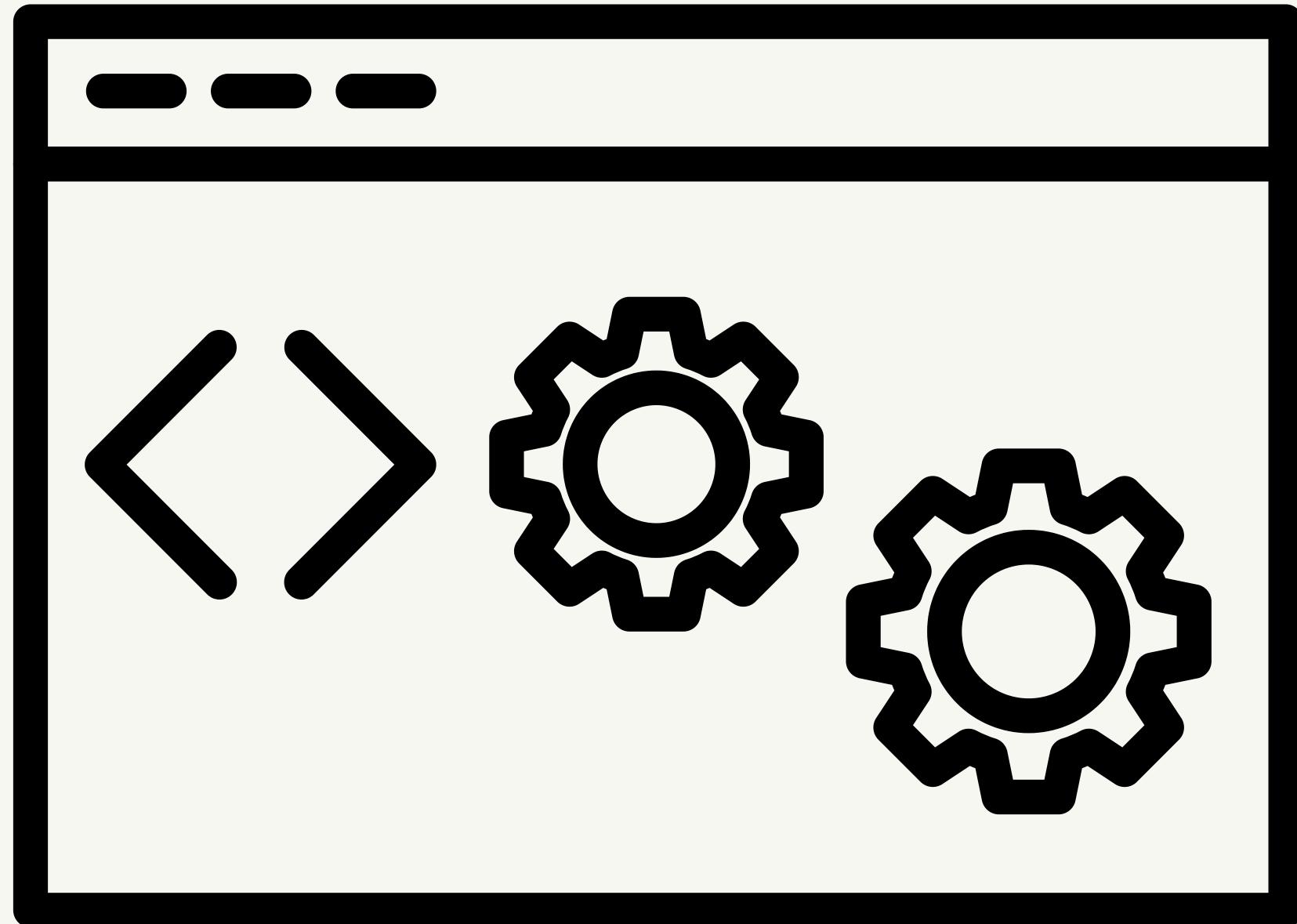
Axis:

<http://{Ip Address}/axis-cgi/jpg/image.cgi?camera={CameraId}&resolution={width}x{height}>

สำหรับยื่ห้อหรือรุ่นอื่นๆเพิ่มเติม [ที่นี่](#)

หา Ip ของกล้องเราให้เจอแล้วลองเรียกดูภาพ จาก url ที่ระบุไว้ผ่าน web browser ดู





# BACKEND

# Unit Test

**API Spec Document**  
**Park Finder**

# PARKFINDER

---

**Documentation**  
**Version 1.0.0**

**{HOST} = 35.240.149.20:5009**

## Login Register Service

### API Register

Description : สำหรับทำการลงทะเบียนผู้ใช้งาน

POST

{HOST}/customer/register

Header

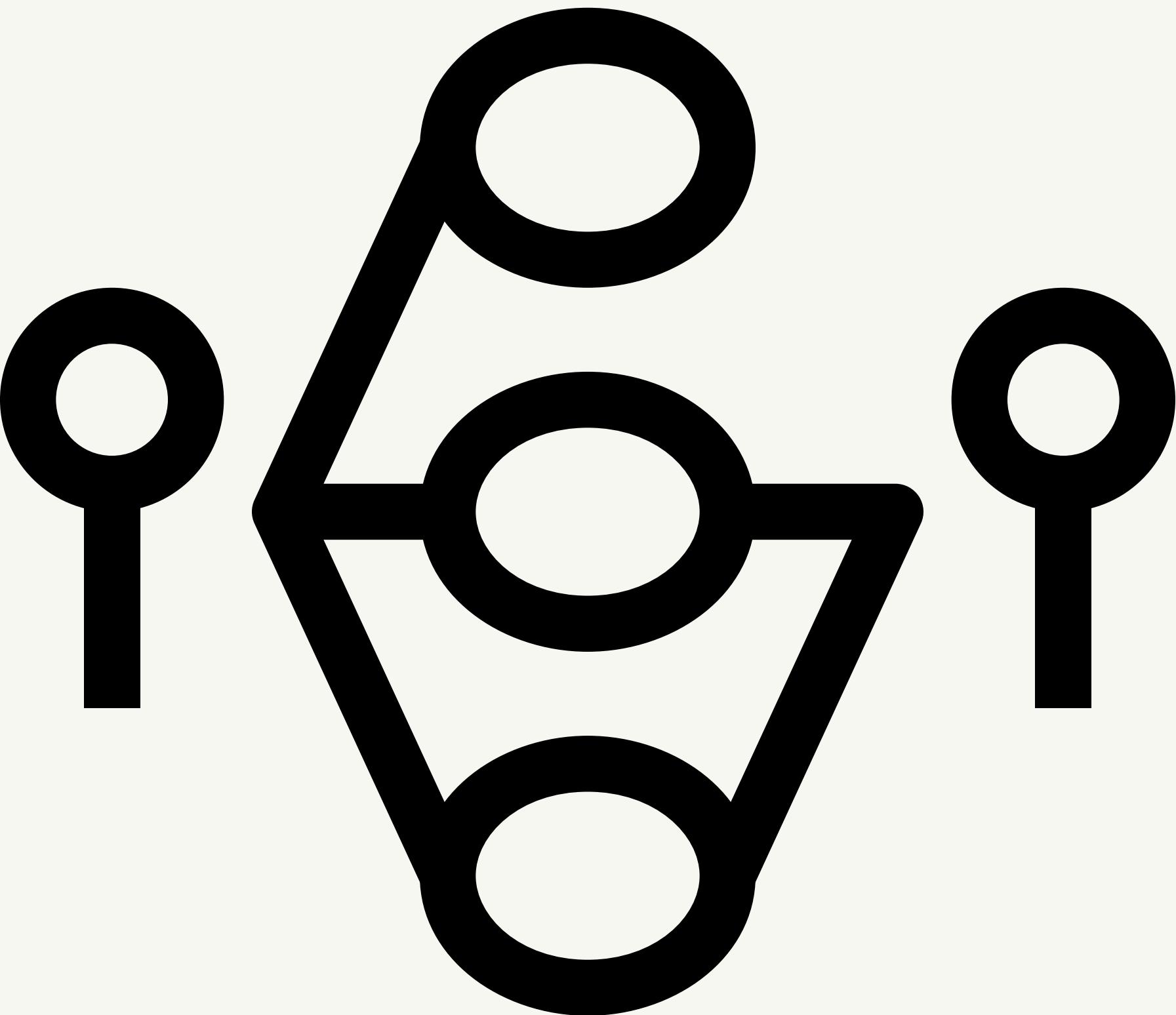
Content-Type: application/json

### Request Body :

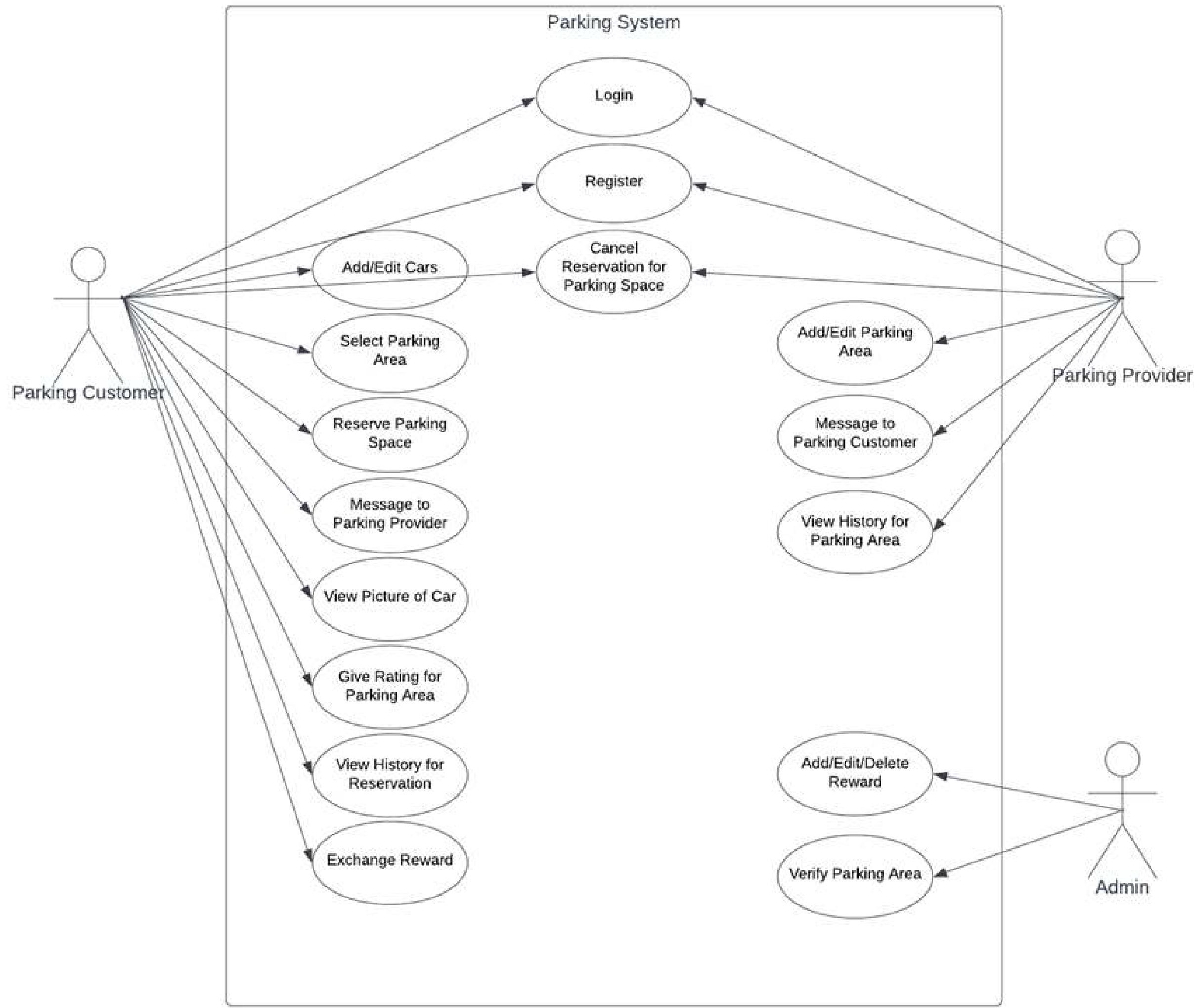
Field name	Data types	Require	Description
first_name	String	Y	ชื่อจริงที่ต้องการลงทะเบียนไว้
last_name	String	Y	นามสกุลที่ต้องการลงทะเบียนไว้
email	String	Y	email ของผู้ใช้งาน
password	String	Y	password ของผู้ใช้งาน

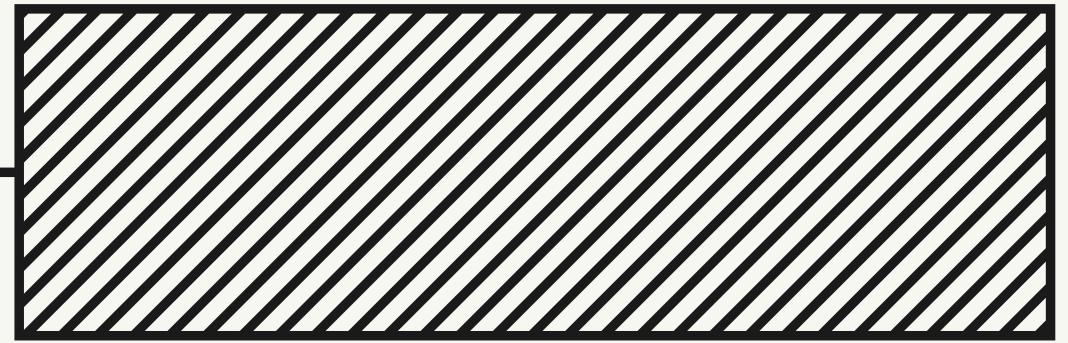
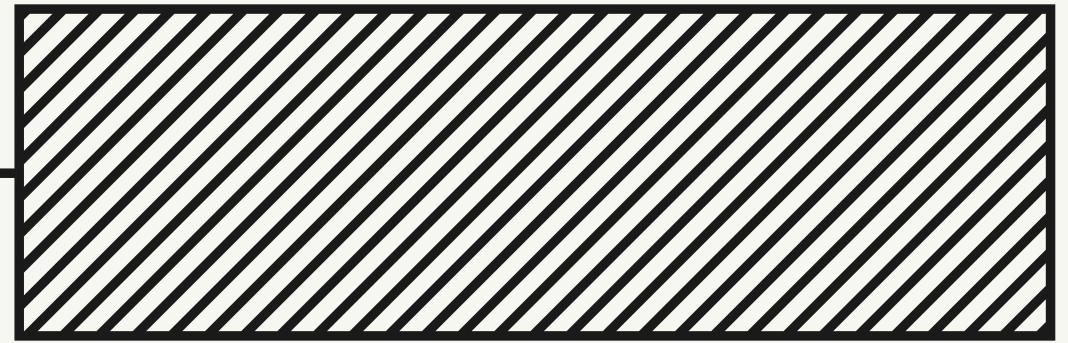
### Body example :

```
{  
    "first_name": "develop",  
    "last_name": "develop",  
    "email": "developer@easyparkfinder.com",  
    "phone": "0804560908",  
    "password": "EasyParkFinder"  
}
```



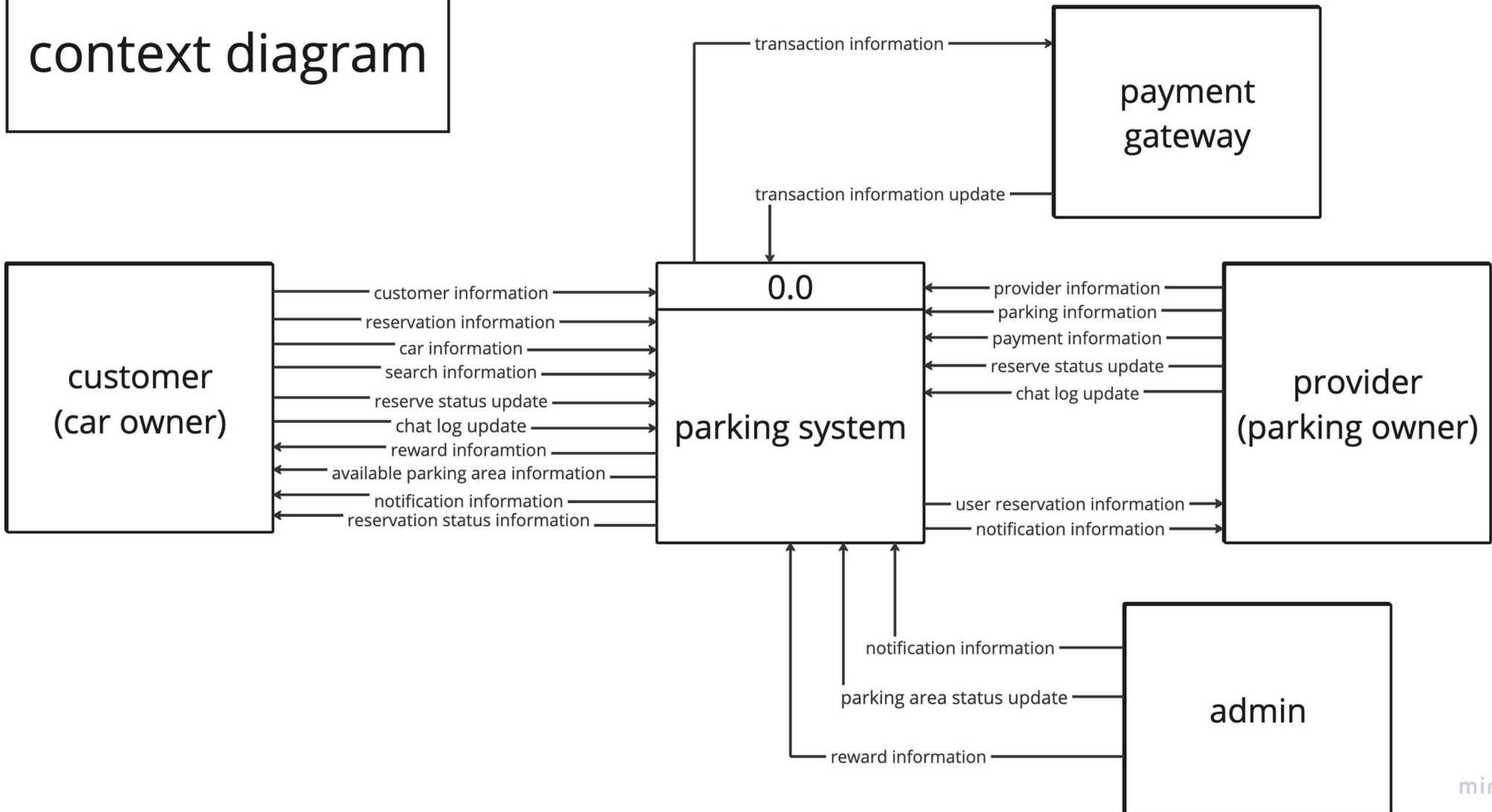
# Use Case Diagram

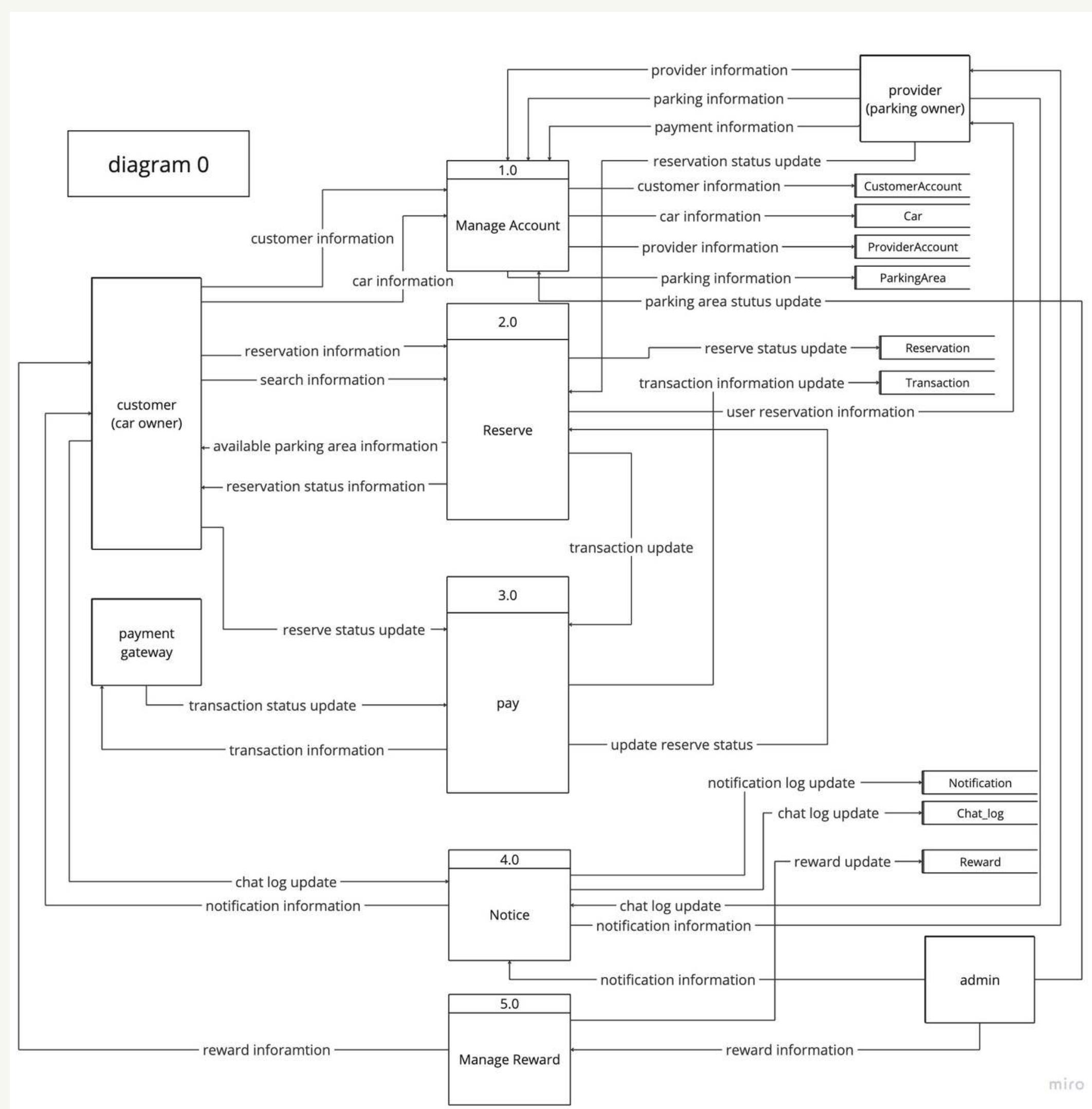


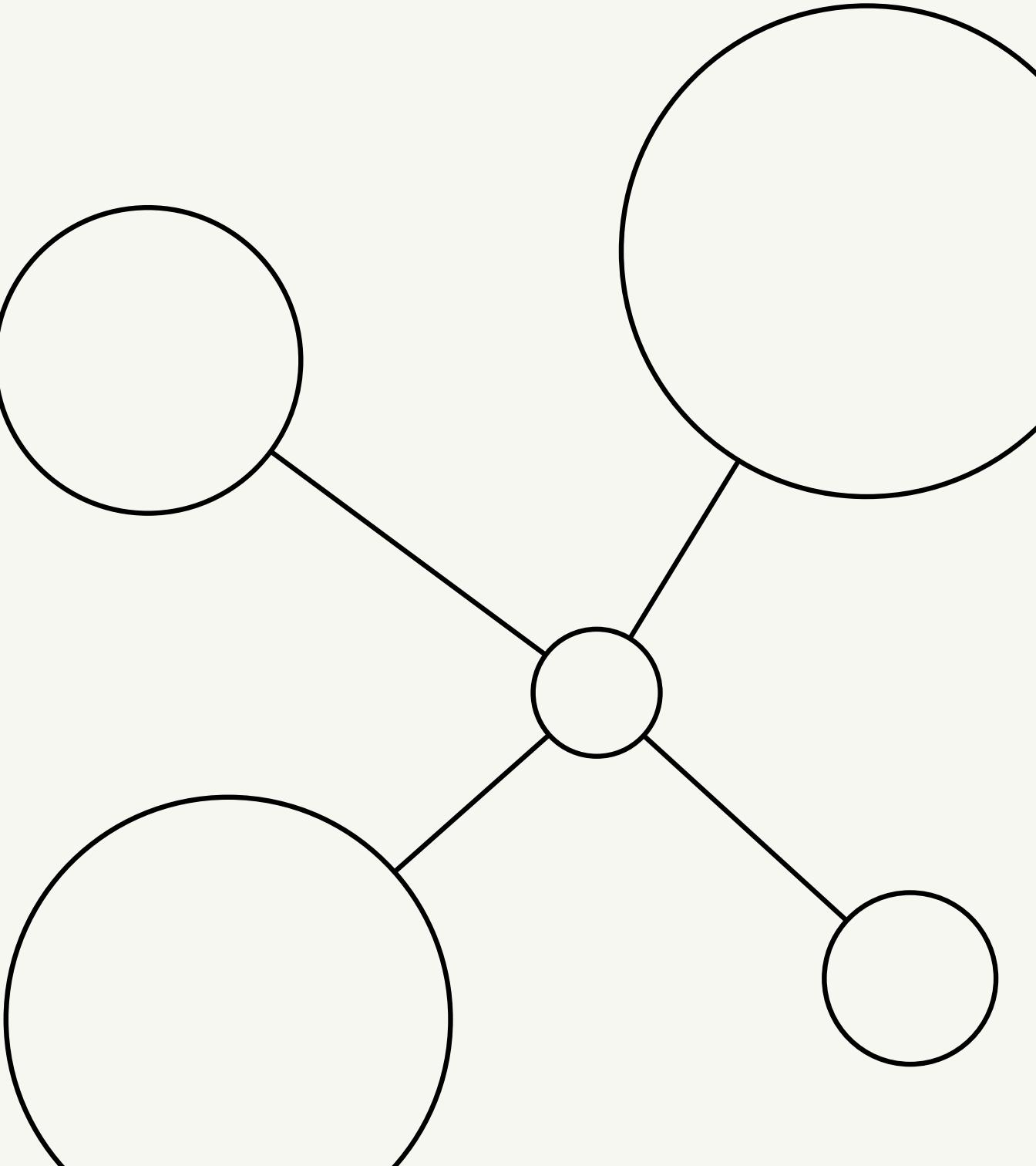
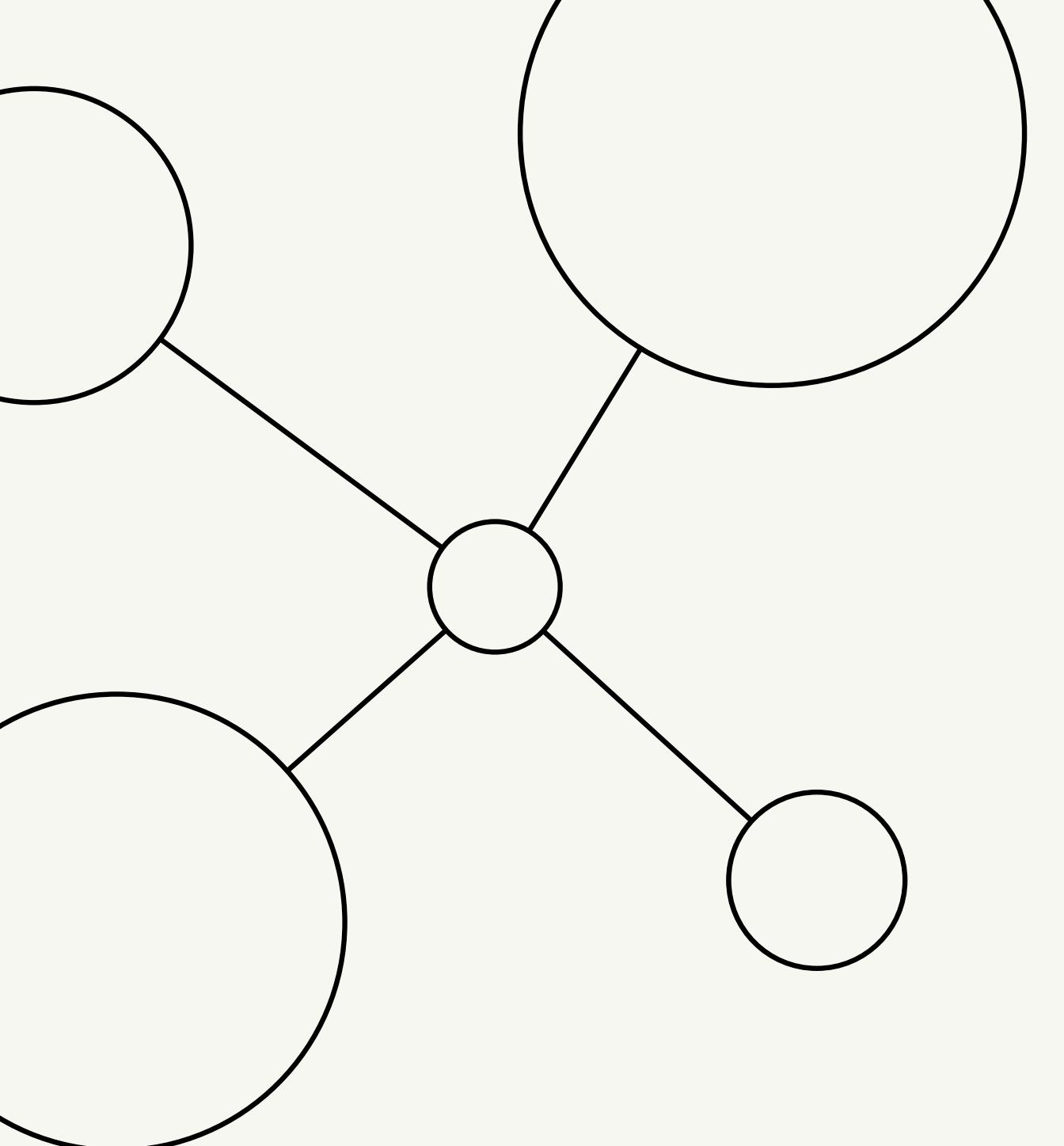


**DFD**

# context diagram

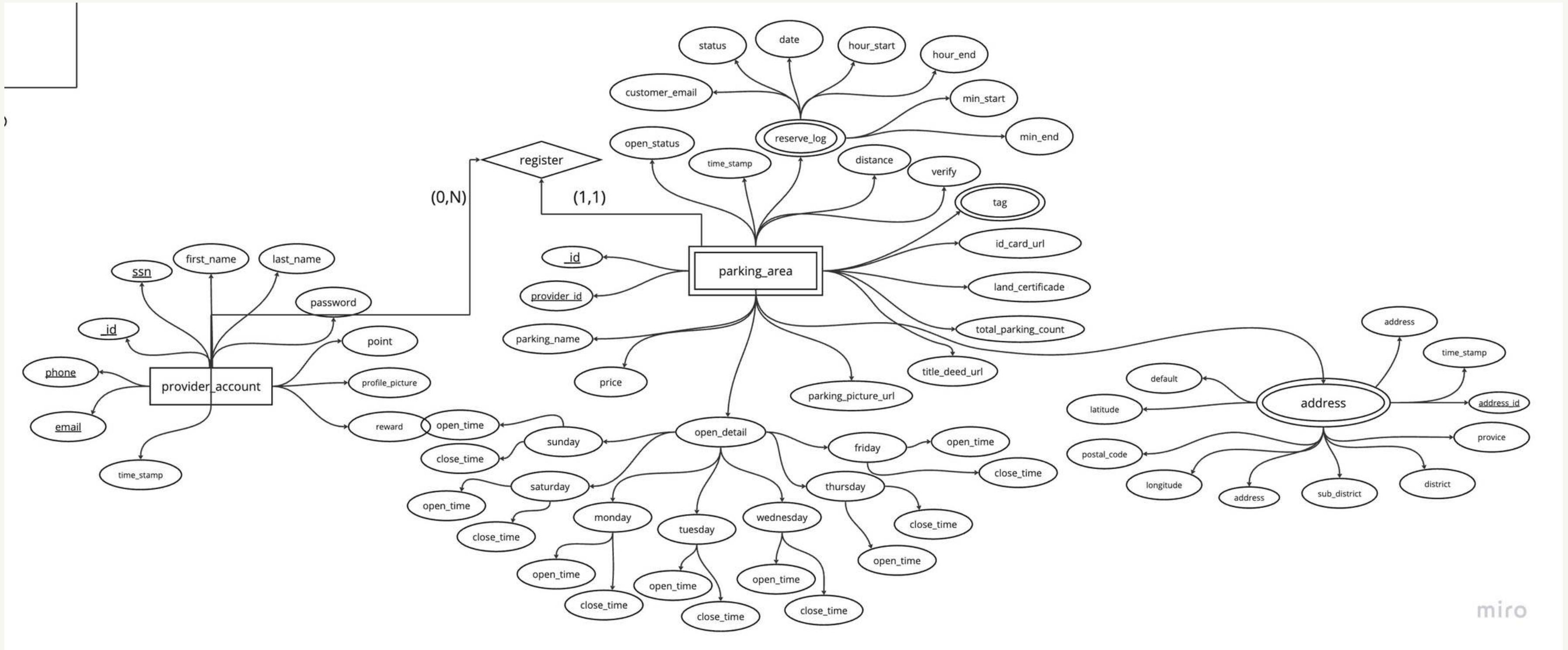




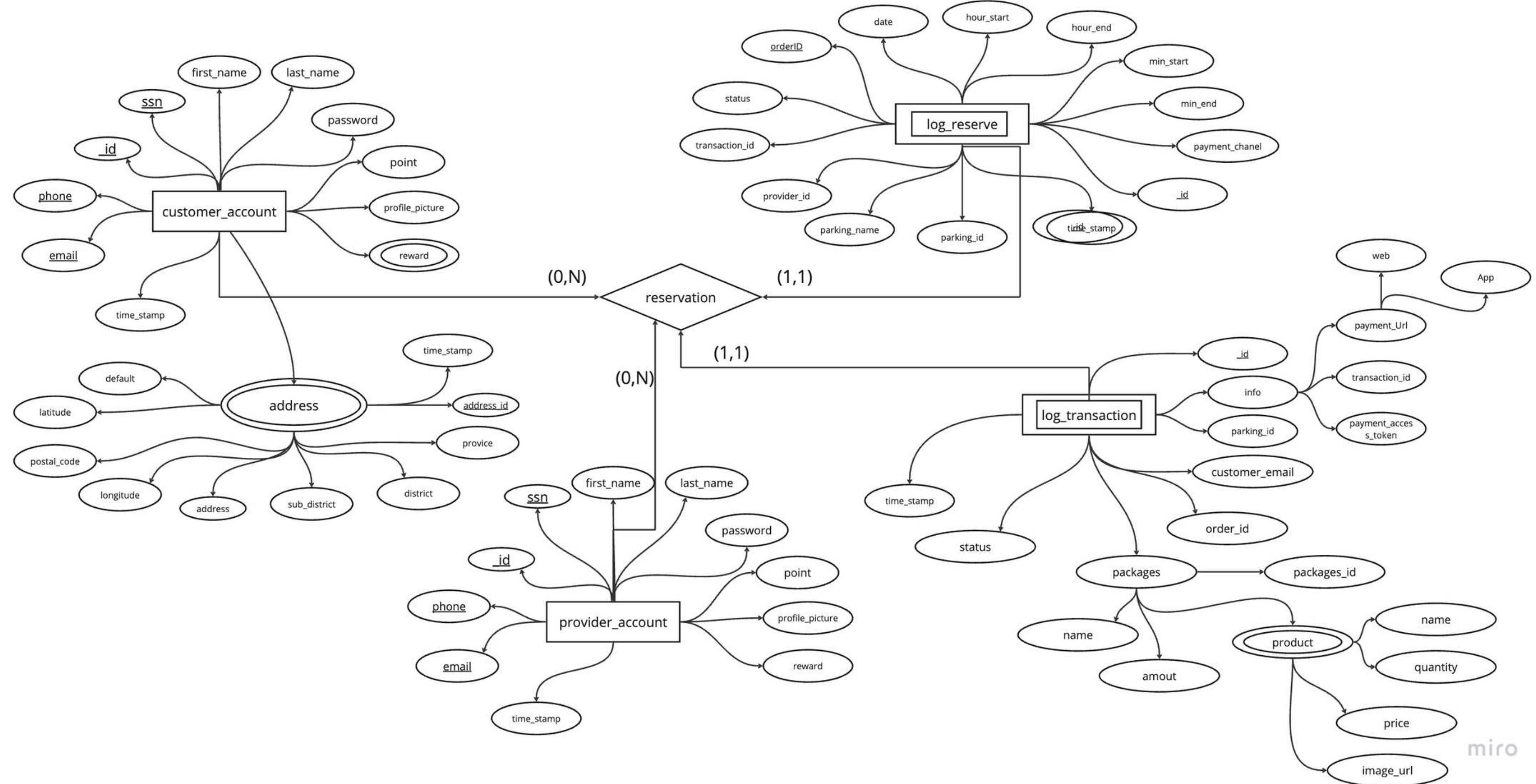


**ER**

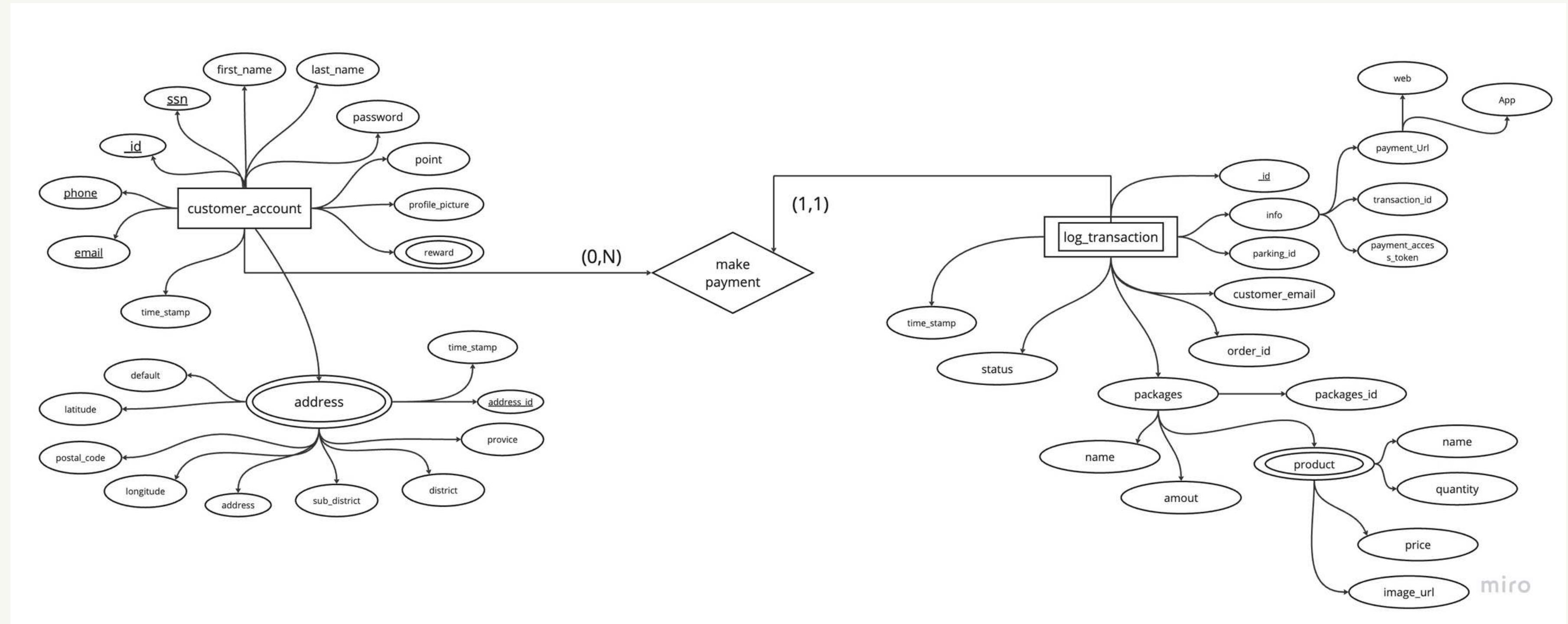
# Register (ລົງທະເບີນກິຈອດຣອ)



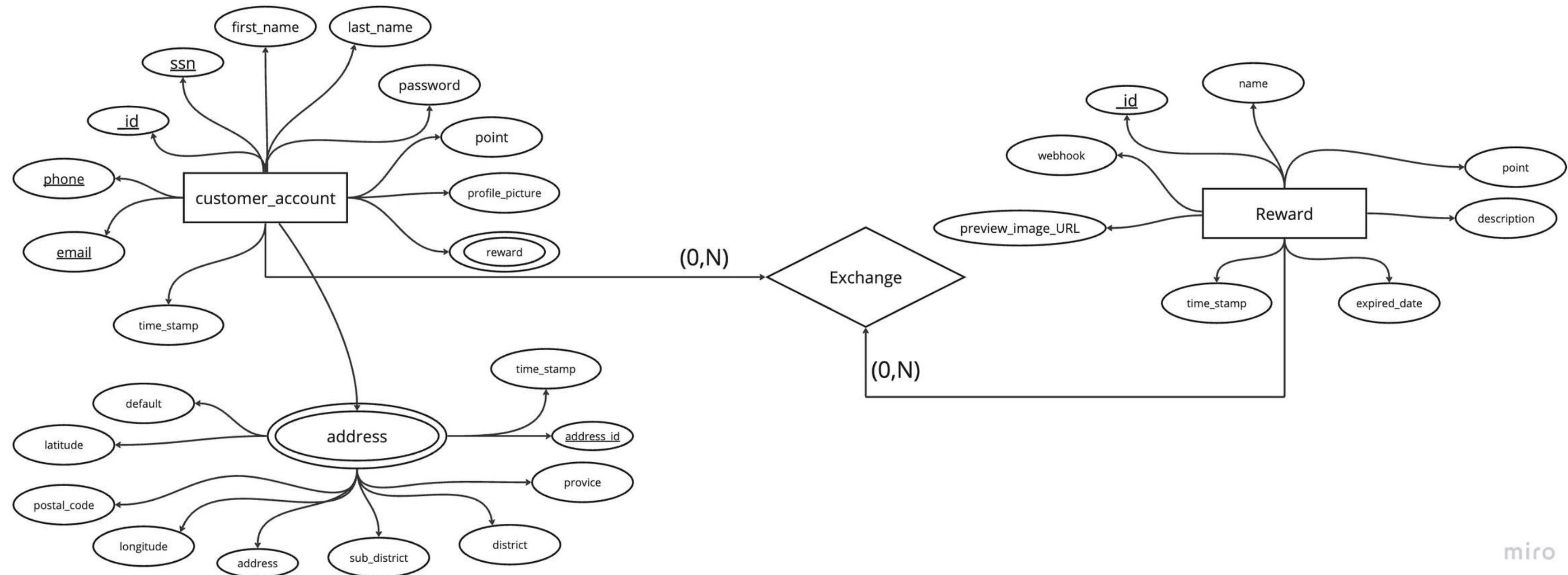
# Reservation (ກາຣຄອງ)



# Make Payment(សរាយ transaction)



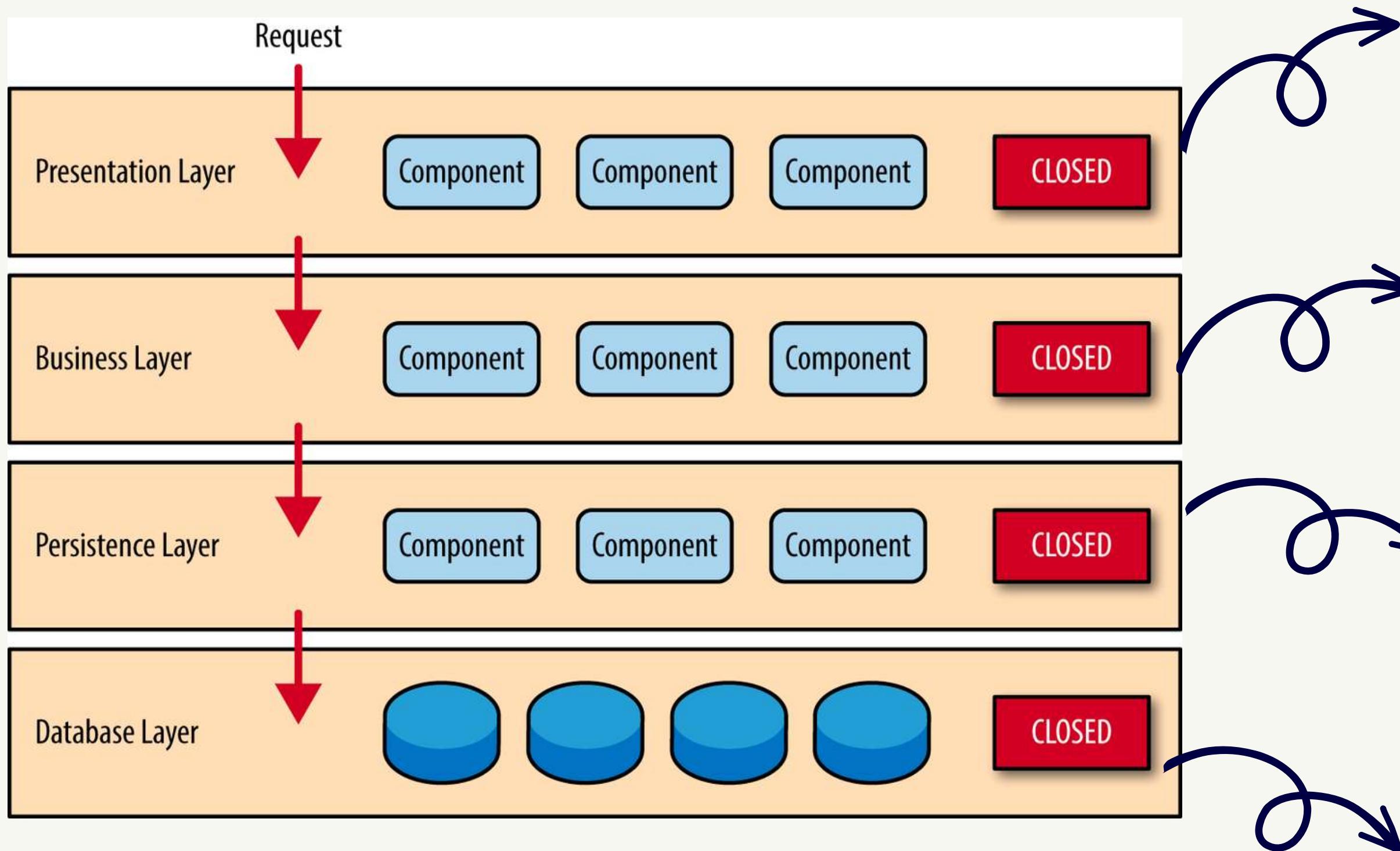
# Exchange(ແລກ Reward)





# Architecture & Design Pattern

# Architecture



```
✓ routers
- go auth_router.go
- go customer_router.go
- go factory.go
- go payment_router.go
- go reserve_router.go
```

```
✓ services
- go auth_service.go
- go customer_service.go
- go factory.go
```

```
✓ storage
- go account_storage.go
- go car_storage.go
- go log_storage.go
- go parking_area_storage.go
- go reward_storage.go
- go token_storage.go
```



# Design Pattern

## Dependency Injection

```
cS := cs.NewCustomerServices(db, redis)
```

```
    result, err := cs.CustomerAccoutStorage.UpdateAccountInterface(ct
```

```
func (cs CustomerServices) SaveOTP(email string, otp string) error {
```

```
    key := fmt.Sprintf(redisOTPKeyFormat, email)
```

```
    // Cache for 10 minute
```

```
    result := cs.Redis.Set(key, otp, 120)
```

```
    err := result.Err()
```

```
    if err != nil {
```

```
        return err
```

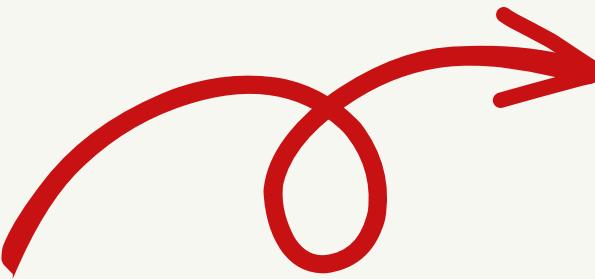
```
}
```

```
    fmt.Println("Cache user otp for email:", email)
```

```
    return nil
```

```
}
```

# Factory Method Pattern



ประกาศไว้ที่ service  
(Business Layer)

```
type ICustomerServices interface {
    CheckExistEmail(ctx context.Context, email string) *models.CustomerAccount
```

```
func (cr CustomerRouters) customerRegisterHandler(c echo.Context) error {
    context := c.Request().Context()

    request := new(models.RegisterAccountRequest)
    if err := c.Bind(request); err != nil {
        return c.JSON(http.StatusBadRequest, models.MessageResponse{Message: err.Error()})
    }

    account := cr.CustomerServices.CheckExistEmail(context, request.Email)
    if account != nil {
        return c.JSON(http.StatusConflict, models.MessageResponse{Message: "This email is already exists"})
    }
}
```

router  
(presentation layer)  
implement  
Customer Service ไว้  
ทำให้สามารถใช้งาน  
ฟังก์ชันของ interface  
Customer Service ได้

# Repository Pattern(1/2)

```
storage
└── account_storage.go
└── car_storage.go
└── log_storage.go
└── parking_area_storage.go
└── reward_storage.go
└── token_strorage.go
```

```
func NewAdminAccoutStorage(db *mongo.Database) *AccoutStorage {
    return &AccoutStorage{
        Collection: db.Collection(os.Getenv("COLLECTION_ADMIN_ACCOUNT_NAME")),
    }
}

func (cs AccoutStorage) InsertAccount(ctx context.Context, data interface{}) (*mongo.InsertOneResult, error) {
    result, err := cs.Collection.InsertOne(ctx, data)
    return result, err
}

func (cs AccoutStorage) UpdateAccountInterface(ctx context.Context, filter interface{}, update interface{}) {
    result, err := cs.Collection.UpdateOne(ctx, filter, update)
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println("Modified count:", result.ModifiedCount)
    return result, err
}
```

# ทำไมถึงใช้ NOSQL

- เนื่องจากตอนออกแบบระบบ คำถึงถึงว่าการระบบของที่จอดรถ ต้องมีการเข้าถึงอัพเดตค่า status ต่าง ๆ อยู่บ่อย ๆ รวมถึงมีระบบสนับนา real-time ซึ่งเข้าใจว่าจะเหมาะสมกว่า
- เคยใช้บ่อยกว่า SQL

