DataCenter2026 Data Center API Manual

Version: Ver1.0 Applies to: Parking_2026_Strandard : DataCenter

Date: 2026-08-01 Reference: COMA Parking API Guidance Ver0.10

0. Overview

This manual describes the HTTP API exposed by the DataCenter2026 data center. On startup DataCenter also starts a built-in HTTP service, so any HTTP-capable client (web page, desktop app, script, third-party platform) can call it over the network to query and manage visit records, VIP customers, charging records, shifts, fee rules and ANPR pictures.

0.1 Conventions

Item	Description
Method	HTTP GET / POST (POST recommended).
Parameters	Passed as a JSON text in the HTTP Body (same as the COMA API). URL query parameters are also accepted.
Format	Request and response are JSON (UTF-8). Suggested header: Content-Type: application/json; charset=utf-8
Response	Uniform: {"success": bool, "message": string, "data": object|array}.
CORS	Enabled: Access-Control-Allow-Origin: *.
Base URL	http://<server-ip>:8200 e.g. http://192.168.1.10:8200/Visit/Query
Default port	8200 (change via ApiPrefix in DataCenter.dll.config).

Auth	No authentication in this version; for trusted networks only.
Time format	yyyy-MM-ddTHH:mm:ss. Use 0001-01-01T00:00:00 when no time filter is needed.
Numeric range	Use -1 for Min/Max and Pay_Type when no range/filter is needed.
Fuzzy search	Fields marked (fuzzy) use LIKE %value%.

0.2 Unified Response

Success:

```
{
    "success": true,
    "message": "Customer Added Successfully",
    "customer_code": "0000000012"
}
```

Failure:

```
{
    "success": false,
    "message": "Already Has A Same Plate"
}
```

List query (empty returns [ ]):

```
{
    "success": true,
    "data": [ { "code": "...", "plate": "..." } ]
}
```

0.3 Startup & Preparation

1) Install the WindowsApp desktop runtime; 2) configure and test the DB connection on the Settings page; 3) start DataCenter - it listens on the HTTP port automatically.

4) Open http://localhost:8200/ to see the full endpoint list.

5) LAN access: ApiPrefix defaults to http://+:8200/. If only localhost works, run once as administrator:

```
netsh http add urlacl url=http://+:8200/ user=Everyone
```

Or run DataCenter as administrator. Without permission the service falls back to localhost only.

Endpoint Index

Method	Path	Description
GET	/	API Index
GET	/Api/Version	Version
GET/POST	/Api/Ping	Ping
GET/POST	/Api/Health	Health Check
POST	/Api/Enum/Query	Query Enum Dictionary
GET	/Api/Enum	Query All Enums
POST/GET	/Camera/List	Camera Configuration
POST/GET	/Info/Get	Get Park Info
POST	/Info/Set	Update Park Info
POST/GET	/Info/Vacant	Vacant Spaces
POST/GET	/Config/Get	Get Config
POST	/Config/Set	Update Config
POST	/Config/TestConnection	Test DB Connection

POST/GET	/Customer/Query	Query VIP Customers
POST/GET	/Customer/Get	Get VIP by Code
POST	/Customer/Add	Add VIP Customer
POST	/Customer/Modify	Modify VIP Customer
POST	/Customer/Delete	Delete VIP Customer
POST	/Customer/Check	Validate VIP Status
POST	/Customer/NextCode	Next Customer Code
POST	/Customer/Statistics	VIP Statistics
POST/GET	/Visit/Query	Query Visit Records
POST/GET	/Visit/Get	Get Visit by Code
POST/GET	/Visit/Ongoing	Ongoing Vehicles
POST	/Visit/Statistics	Visit Statistics
POST	/Visit/Insert	Insert Entry Record
POST	/Visit/CheckOut	Close Visit (Exit)
POST	/Visit/SetOutPicture	Set Exit Picture
POST	/Visit/Delete	Delete Visit Record
POST/GET	/Charge/Query	Query Charge Records
POST	/Charge/Get	Get Charges by Code
POST	/Charge/Insert	Insert Charge Record
POST	/Charge/Delete	Delete Charge Records

POST	/Charge/Statistics	Charge Statistics
POST/GET	/VipCharge/Query	Query VIP Charges
POST	/VipCharge/Get	Get VIP Charge by Id
POST	/VipCharge/Insert	Insert VIP Charge
POST	/VipCharge/Delete	Delete VIP Charge
POST	/VipCharge/Statistics	VIP Charge Statistics
POST/GET	/Shift/Query	Query Shifts
POST	/Shift/Get	Get Shift
POST	/Shift/GroupStats	Shift Pay-type Stats
POST	/Shift/Insert	Open Shift
POST	/Shift/Close	Close Shift
POST	/Shift/Delete	Delete Shift
POST/GET	/Fee/Calculate	Calculate Fee
POST	/Fee/Due	Due Fee
POST	/Fee/Paid	Paid Amount
POST/GET	/Fee/Rules/Get	Get Fee Rules
POST	/Fee/Rules/Set	Set Simple Fee Rules
POST/GET	/Fee/Function/Get	Get Fee Function
POST	/Fee/Function/Set	Set Fee Function
POST	/Fee/LostTicket/Get	Get Lost Ticket Fee

POST	/Fee/LostTicket/Set	Set Lost Ticket Fee
POST	/Fee/LostTicket/Charge	Charge Lost Ticket
POST/GET	/Operator/Query	Query Operators
POST	/Operator/Get	Get Operator
POST	/Operator/Login	Operator Login
POST	/Operator/Add	Add Operator
POST	/Operator/Modify	Modify Operator
POST	/Operator/Delete	Delete Operator
POST/GET	/Picture/Get	Get Picture (Base64)
POST/GET	/Picture/Query	Query Picture Meta
POST	/Picture/Pending	Pending Pictures
POST	/Picture/Insert	Insert Picture
POST	/Picture/SetPlate	Set Plate
POST	/Picture/Delete	Delete Picture
POST/GET	/Report/Combined	Combined Report
POST	/Report/Daily	Daily Summary
POST	/Report/PayTypeSummary	Pay-type Summary
POST	/Report/ShiftSummary	Shift Summary
POST	/Report/OperatorSummary	Operator Summary
POST	/Report/DeviceSummary	Device Summary

POST/GET	/Anpr/Records	ANPR Records
POST	/Anpr/Clear	Clear ANPR Records
POST/GET	/Anpr/Status	ANPR Status

1. System & Index

Service status, endpoint index, enum dictionary and camera configuration.

1. API Index

URL: GET /

Description: Returns service name, version, port, the full endpoint list and usage tips. Open http://<host>:8200/ in a browser.

Response examples:

- OK


```
{"success":true,"data":{"name":"DataCenter2026 API","version":"Ver1.0","port":8200,"endpoint_count":70,"endpoints":[{"method":"POST","path":"/Visit/Query"}]}}
```


2. Version

URL: GET /Api/Version

Description: Returns API name, version and server time.

Response examples:

- OK


```
{"success":true,"data":{"name":"DataCenter2026 API","version":"Ver1.0","server_time":"2026-08-01 10:00:00"}}
```

==========

3. Ping

URL: GET / POST /Api/Ping

Description: Heartbeat; returns pong and server time.

Response examples:

- OK


```
{"success":true,"message":"pong","data":{"server_time":"2026-08-01 10:00:00"}}
```


4. Health Check

URL: GET / POST /Api/Health

Description: Checks the database connection. Returns success=false when the DB is unavailable.

Response examples:

- OK


```
{"success":true,"data":{"database":"Connected","server_time":"2026-08-01 10:00:00","port":8200}}
```


- Error


```
{"success":false,"message":"Database disconnected: ..."}
```


5. Query Enum Dictionary

URL: POST /Api/Enum/Query

Description: Queries the Park_Objects enum dictionary (e.g. Pay_Type).

Request parameters (JSON Body):

Parameter	Type	Required	Description
Object	string	No	Enum name such as Pay_Type. Omit for all.

==========

Request example:

```
{
  "Object": "Pay_Type"
}
```

Response examples:

- OK

```
{"success":true,"data":[{"object":"Pay_Type","value":0,"name":"Cash"}]}
```

6. Query All Enums

URL: GET /Api/Enum

Description: Same as /Api/Enum/Query without parameters.

Response examples:

- OK

```
{"success":true,"data":[{"object":"Pay_Type","value":0,"name":"Cash"}]}
```

7. Camera Configuration

URL: POST / GET /Camera/List

Description: Returns the 6 configured cameras (lanes 0-2 entry, 3-5 exit).

Response examples:

- OK

```
{"success":true,"data":[{"lane":0,"direction":"Entry","ip":"192.168.168.56"}]}
```

2. Park Info (Park_Info)

Read/write park parameters: total spaces, exit time limit, lost ticket fee.

==========

8. Get Park Info

URL: POST / GET /Info/Get

Description: Returns Total_Spaces, Limited_Exit_Time (minutes), Lost_Ticket_Fee and Simple_Fee_Rules. Returns defaults if the table has no row.

Response examples:

- OK


```
{"success":true,"data":{"total_spaces":1000,"limited_exit_time":15,"lost_ticket_fee":500.00,"simple_fee_rules":"[...]"}}
```


9. Update Park Info

URL: POST /Info/Set

Description: Only the provided fields are updated.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Total_Spaces	int	No	Total parking spaces
Limited_Exit_Time	int	No	Exit time limit (minutes)
Lost_Ticket_Fee	number	No	Lost ticket fee

Request example:

```
{
  "Total_Spaces": 1000,
  "Limited_Exit_Time": 15,
  "Lost_Ticket_Fee": 500
}
```

Response examples:

==========

- OK


```
{"success":true,"message":"Park_Info updated successfully"}
```


- Error


```
{"success":false,"message":"No fields to update."}
```


10. Vacant Spaces

URL: POST / GET /Info/Vacant

Description: Vacant = total spaces - open parking records.

Response examples:

- OK


```
{"success":true,"data":{"total_spaces":1000,"used_spaces":123,"vacant_spaces":877,"server_time":"2026-08-01 10:00:00"}}
```


3. Connection & Runtime Config

Read/modify the database connection and the API listener.

11. Get Config

URL: POST / GET /Config/Get

Description: Returns SQL host, user, catalog (password masked), API prefix and port.

Response examples:

- OK


```
{"success":true,"data":{"sql_host":"localhost","sql_user_name":"sa","sql_password":"******","sql_catalog":"Parking_2026_Strandard","api_prefix":"http://+:8200/","api_port":8200,"api_running":true}}
```

==========

12. Update Config

URL: POST /Config/Set

Description: Writes the DB connection or API prefix. An ApiPrefix change needs a restart.

Request parameters (JSON Body):

Parameter	Type	Required	Description
SQLHost	string	No	SQL host
SQLUserName	string	No	SQL user
SQLPassword	string	No	SQL password
SQLCatalog	string	No	Database name
ApiPrefix	string	No	Listener prefix e.g. http://+:8200/

Request example:

```
{
  "SQLHost": "192.168.1.10",
  "SQLUserName": "sa",
  "SQLPassword": "8210",
  "SQLCatalog": "Parking_2026_Strandard"
}
```

Response examples:

- OK

```
{"success":true,"message":"Configuration saved. Restart may be required for ApiPrefix change."}
```

13. Test DB Connection

URL: POST /Config/TestConnection

Description: Connects using the current config and runs SELECT 1.

==========

Response examples:

- OK

```
{"success":true,"message":"Database connection succeeded"}
```

- Error

```
{"success":false,"message":"Database error: ..."}
```

4. VIP Customer (Park_Customer)

Manage monthly/VIP customers: query, add, modify, delete, validate, statistics.

14. Query VIP Customers

URL: POST / GET /Customer/Query

Description: Fuzzy match on Customer_Code, Card_Id, Name, Phone, Plate; date ranges on registration/expiry. Returns success=false when empty.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	No	Customer code (fuzzy)
Card_Id	string	No	Card id (fuzzy)
Name	string	No	Name (fuzzy)
Phone	string	No	Phone (fuzzy)
Plate	string	No	Plate (fuzzy)
Reg_Dt_start	datetime	No	Registration from
Reg_Dt_end	datetime	No	Registration to

Valid_dt_start	datetime	No	Expiry from
Valid_dt_end	datetime	No	Expiry to

Request example:

```
{
  "Plate": "T367",
  "Valid_dt_start": "0001-01-01T00:00:00",
  "Valid_dt_end": "0001-01-01T00:00:00"
}
```

Response examples:

- OK

```
{"success":true,"data":[{"customer_code":"0000000001","card_id":"0007623998","name":"DIRECTOR","phone":"0754262040","plate":"T367AYG","reg_dt":"2025-09-01     00:00:00","expire_dt":"2025-12-31 00:00:00","is_expired":false}]}
```

- Empty

```
{"success":false,"message":"No VIP Users Found In Such Selector"}
```

15. Get VIP by Code

URL: POST / GET /Customer/Get

Description: Exact read by Customer_Code.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Customer_Code	string	Yes	Customer code

Request example:

```
{
```

==========

```
  "Customer_Code": "0000000001"
}
```

Response examples:

- OK

```
{"success":true,"data":{...}}
```

- Error

```
{"success":false,"message":"Customer not found"}
```

16. Add VIP Customer

URL: POST /Customer/Add

Description: Adds a monthly customer. Card_Id required; Customer_Code auto-generated when omitted. Expiry must be after registration.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Customer_Code	string	No	Customer code (auto if omitted)
Card_Id	string	Yes	Card id
Name	string	No	Name
Phone	string	No	Phone
Plate	string	No	Plate
Reg_Dt	datetime	No	Registration date (default now)
Valid_dt	datetime	No	Expiry date (default +1 month)

==========

Request example:

```
{
  "Name": "DIRECTOR",
  "Plate": "T367AYG",
  "Phone": "0754262040",
  "Card_Id": "0007623998",
  "Valid_dt": "2026-12-31T00:00:00"
}
```

Response examples:

- OK

```
{"success":true,"message":"Customer Added Successfully","customer_code":"0000000012"}
```

- Dup plate

```
{"success":false,"message":"Already Has A Same Plate"}
```

- Dup card

```
{"success":false,"message":"Already Has A Same Card_id"}
```

- Missing

```
{"success":false,"message":"Card_Id is required."}
```

17. Modify VIP Customer

URL: POST /Customer/Modify

Description: Customer_Code required; only provided fields are updated.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Customer_Code	string	Yes	Customer code

Card_Id	string	No	Card id
Name	string	No	Name
Phone	string	No	Phone
Plate	string	No	Plate
Valid_dt	datetime	No	Expiry date

Request example:

```
{
  "Customer_Code": "0000000012",
  "Phone": "13800000000"
}
```

Response examples:

- OK

```
{"success":true,"message":"Customer information updated successfully"}
```

- No change

```
{"success":false,"message":"No changes to update. If need to remove users, USE /Customer/Delete"}
```

- Not found

```
{"success":false,"message":"Customer not found"}
```

18. Delete VIP Customer

URL: POST /Customer/Delete

Description: Delete by Customer_Code.

Request parameters (JSON Body):

==========

Parameter	Type	Required	Description
Customer_Code	string	Yes	Customer code

Request example:

```
{
  "Customer_Code": "0000000012"
}
```

Response examples:

- OK

```
{"success":true,"message":"Customer deleted successfully"}
```

- Error

```
{"success":false,"message":"Customer not found"}
```

19. Validate VIP Status

URL: POST /Customer/Check

Description: Look up by Plate / Card_Id / Customer_Code; returns expired, has open record, can enter.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Plate	string	No	Plate
Card_Id	string	No	Card id
Customer_Code	string	No	Customer code

Request example:

```
{
```

==========

```
  "Plate": "T367AYG"
}
```

Response examples:

- OK

```
{"success":true,"data":{"customer":{...},"is_expired":false,"has_open_record":false,"can_enter":true}}
```

- Error

```
{"success":false,"message":"No Such VIP User"}
```

20. Next Customer Code

URL: POST /Customer/NextCode

Description: Returns the next available 10-digit customer code.

Response examples:

- OK

```
{"success":true,"data":{"customer_code":"0000000013"}}
```

21. VIP Statistics

URL: POST /Customer/Statistics

Description: Total, valid and expired VIP count (optional registration date filter).

Request parameters (JSON Body):

Parameter	Type	Required	Description
Reg_Dt_start	datetime	No	Registration from
Reg_Dt_end	datetime	No	Registration to

==========

Response examples:

- OK

```
{"success":true,"data":{"total":100,"valid":60,"expired":40}}
```

5. Visit Records (Park_Records)

Query vehicle entry/exit records; insert and close records.

22. Query Visit Records

URL: POST / GET /Visit/Query

Description: Fuzzy match on Code/Plate/Ticket_Id/Card_Id/Customer_Code/In_Device/Out_Device; in/out time ranges; State filter.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	No	Record code (fuzzy)
Plate	string	No	Plate (fuzzy)
Ticket_Id	string	No	Ticket id (fuzzy)
Card_Id	string	No	Card id (fuzzy)
Customer_Code	string	No	Customer code (fuzzy)
In_Device	string	No	Entry device (fuzzy)
Out_Device	string	No	Exit device (fuzzy)
in_time_start	datetime	No	Entry time from
in_time_end	datetime	No	Entry time to

out_time_start	datetime	No	Exit time from
out_time_end	datetime	No	Exit time to
State	string	No	Ongoing = in lot, Closed = exited
Top	int	No	Return top N rows

Request example:

```
{
  "Plate": "T367AYG",
  "in_time_start": "0001-01-01T00:00:00",
  "in_time_end": "0001-01-01T00:00:00"
}
```

Response examples:

- OK


```
{"success":true,"data":[{"code":"P20260801130311346","plate":"T367AYG","ticket_id":"","card_id":
"","customer_code":"0000000001","in_device":"ENTRY#1","in_time":"2026-08-01
13:03:11","in_picture_id":101,"out_device":"","out_time":null,"out_picture_id":0}]}
```


23. Get Visit by Code

URL: POST / GET /Visit/Get

Description: Exact read by Code.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code

Request example:

```
{
```

==========

```
  "Code": "P20260801130311346"
}
```

Response examples:

- OK

```
{"success":true,"data":{...}}
```

- Error

```
{"success":false,"message":"Code Not Exist"}
```

24. Ongoing Vehicles

URL: POST / GET /Visit/Ongoing

Description: Records with Out_Time IS NULL (still in lot), optional fuzzy Plate/Card_Id.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Plate	string	No	Plate (fuzzy)
Card_Id	string	No	Card id (fuzzy)

Request example:

```
{
  "Plate": "T367"
}
```

Response examples:

- OK

```
{"success":true,"data":[...]}
```

==========

25. Visit Statistics

URL: POST /Visit/Statistics

Description: Total, ongoing and closed counts (optional entry time filter).

Request parameters (JSON Body):

Parameter	Type	Required	Description
in_time_start	datetime	No	Entry time from
in_time_end	datetime	No	Entry time to

Response examples:

- OK


```
{"success":true,"data":{"total":500,"ongoing":20,"closed":480}}
```


26. Insert Entry Record

URL: POST /Visit/Insert

Description: Manually insert an entry record. At least one of Plate / Ticket_Id / Card_Id; Code auto-generated if omitted.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	No	Record code (auto)
Plate	string	No	Plate
Ticket_Id	string	No	Ticket id
Card_Id	string	No	Card id

Cust_Code	string	No	Customer code
In_Device	string	No	Entry device (default DataCenter)
In_Time	datetime	No	Entry time (default now)
In_Picture_Id	long	No	Entry picture id

Request example:

```
{
  "Plate": "STN2171",
  "In_Device": "ENTRY#1"
}
```

Response examples:

- OK

```
{"success":true,"message":"Car In successfully","data":{"code":"P20260801130311346"}}
```

- Error

```
{"success":false,"message":"At least one of Plate / Ticket_Id / Card_Id is required."}
```

27. Close Visit (Exit)

URL: POST /Visit/CheckOut

Description: By Code, writes out time and exit device. Commit=false checks only.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code
Out_Device	string	No	Exit device (default DataCenter)

Commit	bool	No	true=write exit (default), false=check only

Request example:

```
{
  "Code": "P20260801130311346",
  "Out_Device": "EXIT#1"
}
```

Response examples:

- OK

```
{"success":true,"message":"Exit Success","data":{"code":"P20260801130311346"}}
```

- Already

```
{"success":false,"message":"Already Exited"}
```

- Error

```
{"success":false,"message":"Code Not Exist"}
```

28. Set Exit Picture

URL: POST /Visit/SetOutPicture

Description: By Code, sets the exit picture id.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code
Out_Picture_Id	long	Yes	Exit picture id

Request example:

==========

```
{
  "Code": "P20260801130311346",
  "Out_Picture_Id": 205
}
```

Response examples:

- OK

```
{"success":true,"message":"Out picture updated successfully"}
```

29. Delete Visit Record

URL: POST /Visit/Delete

Description: Delete by Code (use with care).

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code

Request example:

```
{
  "Code": "P20260801130311346"
}
```

Response examples:

- OK

```
{"success":true,"message":"Record deleted successfully"}
```

- Error

```
{"success":false,"message":"Record not found"}
```

==========

6. Parking Charging Records (Park_ParkingChargingRecords)

Query/write parking charge records (simulate cashier) and statistics.

30. Query Charge Records

URL: POST / GET /Charge/Query

Description: Fuzzy Code/Operator/Machine; charge time range, fee range, Pay_Type, Shift_Id, lost-ticket filter.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	No	Record code (fuzzy)
Operator	string	No	Cashier (fuzzy)
Machine	string	No	Machine (fuzzy)
Charge_Dt_start	datetime	No	Charge time from
Charge_Dt_end	datetime	No	Charge time to
Fee_Amount_Min	number	No	Fee min (-1 = none)
Fee_Amount_Max	number	No	Fee max (-1 = none)
Pay_Type	int	No	Pay type (-1 = none)
Shift_Id	int	No	Shift id (-1 = none)
Is_Lost_Ticket	bool	No	Lost ticket

Request example:

```
{
  "Fee_Amount_Min": -1,
```

==========

```
  "Fee_Amount_Max": -1,
  "Pay_Type": -1
}
```

Response examples:

- OK

```
{"success":true,"data":[{"code":"P...","fee":20.00,"pay_type":0,"pay_type_name":"Cash","charge_d
t":"2026-08-01
14:00:00","machine":"Casher0","operator":"admin","shift_id":3,"is_lost_ticket":false}]}
```

31. Get Charges by Code

URL: POST /Charge/Get

Description: All charge records of the given Code.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code

Request example:

```
{
  "Code": "P20260801130311346"
}
```

Response examples:

- OK

```
{"success":true,"data":[...]}
```

- Error

```
{"success":false,"message":"Code Not Exist"}
```

==========

32. Insert Charge Record

URL: POST /Charge/Insert

Description: Simulate a cashier write. Code must exist; Fee, Pay_Type and User_id required.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code
Fee	number	Yes	Fee amount
Pay_Type	int	Yes	Pay type
User_id	string	Yes	Cashier (Operator)
Machine	string	No	Machine (default DataCenter)
Shift_Id	int	No	Shift id
Is_Lost_Ticket	bool	No	Lost ticket
Charge_Dt	datetime	No	Charge time (default now)

Request example:

```
{
  "Code": "P20260801130311346",
  "Fee": 210,
  "Pay_Type": 0,
  "User_id": "admin"
}
```

Response examples:

- OK

```
{"success":true,"message":"Success","data":{"code":"P...","fee":210,"selector":"Code"}}
```

- No fee

```
{"success":false,"message":"Fee Value Lacked Or Incorrect Or Json Body Incorrect"}
```

- No pay

```
{"success":false,"message":"Pay_Type Value Lacked Or Incorrect"}
```

- No user

```
{"success":false,"message":"User_id Value Lacked"}
```

- Not found

```
{"success":false,"message":"Code Not Exist"}
```

33. Delete Charge Records

URL: POST /Charge/Delete

Description: Delete by Code (Confirm=true); optional exact Charge_Dt.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code
Charge_Dt	datetime	No	Exact charge time
Confirm	bool	Yes	Must be true

Request example:

```
{
  "Code": "P20260801130311346",
  "Confirm": true
}
```

==========

Response examples:

- OK

```
{"success":true,"message":"Deleted 1 charging record(s)"}
```

- Error

```
{"success":false,"message":"Set Confirm=true to delete charging records of this Code"}
```

34. Charge Statistics

`URL: POST  /Charge/Statistics`

Description: Count, total fee, lost count and amount.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Charge_Dt_start	datetime	No	Charge time from
Charge_Dt_end	datetime	No	Charge time to
Pay_Type	int	No	Pay type (-1 = none)

Response examples:

- OK

```
{"success":true,"data":{"count":300,"total_fee":12000.00,"lost_count":2,"lost_amount":1000.00}}
```

7. VIP Charging Records (Park_VIPChargingRecords)

Query/write VIP opening and renewal charges.

==========

35. Query VIP Charges

URL: POST / GET /VipCharge/Query

Description: Fuzzy Cust_Code/Card_Id/Name/Machine/Operator; Operation, charge time, validity range, amount, Pay_Type filters.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Cust_Code	string	No	Customer code (fuzzy)
Card_Id	string	No	Card id (fuzzy)
Name	string	No	Name (fuzzy)
Operation	string	No	Add / Extend
Fee_dt_start	datetime	No	Charge time from
Fee_dt_end	datetime	No	Charge time to
Period_start	datetime	No	Validity from
Period_end	datetime	No	Validity to
Fee_Amount_Min	number	No	Fee min (-1 = none)
Fee_Amount_Max	number	No	Fee max (-1 = none)
Pay_Type	int	No	Pay type (-1 = none)

Request example:

```
{
  "Operation": "Add",
  "Fee_Amount_Min": -1,
  "Fee_Amount_Max": -1
```

==========

```
}
```

Response examples:

- OK


```
{"success":true,"data":[{"id":1,"customer_code":"0000000001","card_id":"0007623998","name":"DIRE
CTOR","operation":"Add","operation_name":"New                    VIP","period_start":"2025-09-01
00:00:00","period_end":"2025-12-31
00:00:00","days":121,"rate":8.26,"fee":1000.00,"pay_type":0,"pay_type_name":"Cash","charge_dt":"
2025-09-01 10:00:00","machine":"DataCenter","operator":"API"}]}
```


36. Get VIP Charge by Id

URL: POST /VipCharge/Get

Description: Read one VIP charge by Id.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	long	Yes	Record id

Request example:

```
{
  "Id": 1
}
```

Response examples:

- OK

```
{"success":true,"data":{...}}
```

- Error

```
{"success":false,"message":"Record Not Found"}
```

==========

37. Insert VIP Charge

URL: POST /VipCharge/Insert

Description: At least Cust_Code or Card_Id; Days/Rate auto-derived when omitted.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Cust_Code	string	No	Customer code
Card_Id	string	No	Card id
Name	string	No	Name
Operation	string	No	Add / Extend (default Add)
Period_Start	datetime	No	Validity from (default now)
Period_End	datetime	No	Validity to (default +1 month)
Days	int	No	Days (auto)
Rate	number	No	Daily rate (default Fee/Days)
Fee	number	No	Amount
Pay_Type	int	No	Pay type
Machine	string	No	Machine

Request example:

```
{
  "Cust_Code": "0000000001",
  "Operation": "Extend",
  "Period_Start": "2026-01-01T00:00:00",
  "Period_End": "2026-04-01T00:00:00",
  "Fee": 900,
```

==========

```
  "Pay_Type": 0
}
```

Response examples:

- OK

```
{"success":true,"message":"VIP charging record added successfully"}
```

38. Delete VIP Charge

URL: POST /VipCharge/Delete

Description: Delete by Id, Confirm=true.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	long	Yes	Record id
Confirm	bool	Yes	Must be true

Request example:

```
{
  "Id": 5,
  "Confirm": true
}
```

Response examples:

- OK

```
{"success":true,"message":"VIP charging record deleted successfully"}
```

39. VIP Charge Statistics

URL: POST /VipCharge/Statistics

==========

Description: Count, total fee and total days.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Charge_Dt_start	datetime	No	Charge time from
Charge_Dt_end	datetime	No	Charge time to
Operation	string	No	Add / Extend

Response examples:

- OK


```
{"success":true,"data":{"count":50,"total_fee":45000.00,"total_days":6000}}
```


8. Shifts (Park_Shifts)

Query shifts and their live charge totals, open/close/delete. NOTE: shift features require the DB script Scripts\Update_Shift.sql (adds Park_Shifts table and Shift_Id / Is_Lost_Ticket columns). Until applied, these endpoints and shift-related reports return a clear message.

40. Query Shifts

`URL: POST / GET  /Shift/Query`

Description: Fuzzy Machine/Operator, status, start/end time ranges; live-charge summary per shift.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Machine	string	No	Machine (fuzzy)

Operator	string	No	Operator (fuzzy)
Status	int	No	0=Open, 1=Closed, other=All
Start_Dt_start	datetime	No	Start time from
Start_Dt_end	datetime	No	Start time to
End_time_start	datetime	No	End time from
End_time_end	datetime	No	End time to

Request example:

```
{
  "Machine": "Casher",
  "Status": -1
}
```

Response examples:

- OK

```
{"success":true,"data":[{"id":3,"machine":"Casher0","operator":"admin","start_dt":"2026-08-01 08:00:00","end_dt":null,"status":0,"status_text":"Open","charge_count":12,"total_fee":560.00,"average_fee":46.67,"lost_count":0,"lost_amount":0.00}]}
```

- No table

```
{"success":false,"message":"Park_Shifts table not found. Please run Scripts\Update_Shift.sql to enable shift features."}
```

41. Get Shift

URL: POST /Shift/Get

Description: Read one shift plus per-pay-type groups.

Request parameters (JSON Body):

==========

Parameter	Type	Required	Description
Id	int	Yes	Shift id

Request example:

```
{
  "Id": 3
}
```

Response examples:

- OK


```
{"success":true,"data":{"id":3,"machine":"Casher0","charge_count":12,"total_fee":560.00,"by_pay_type":[{"pay_type":0,"pay_type_name":"Cash","count":10,"fee":500.00}]}}
```


42. Shift Pay-type Stats

URL: POST /Shift/GroupStats

Description: Per-pay-type counts and fees for a shift.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	int	Yes	Shift id

Request example:

```
{
  "Id": 3
}
```

Response examples:

- OK

==========

{"success":true,"data":[{"pay_type":0,"pay_type_name":"Cash","count":10,"fee":500.00}]}

43. Open Shift

URL: POST /Shift/Insert

Description: Opens a new (in-progress) shift. A machine cannot have two open shifts.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Machine	string	No	Machine (default DataCenter)
Operator	string	No	Operator (default API)

Request example:

```
{
  "Machine": "Casher0",
  "Operator": "admin"
}
```

Response examples:

- OK

{"success":true,"message":"Shift opened successfully","data":{"id":4,"machine":"Casher0","operator":"admin"}}

- Conflict

{"success":false,"message":"This machine already has an open shift."}

44. Close Shift

URL: POST /Shift/Close

Description: Close by Id; writes end time and snapshots totals.

==========

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	int	Yes	Shift id

Request example:

```
{
  "Id": 4
}
```

Response examples:

- OK

```
{"success":true,"message":"Shift closed successfully","data":{"id":4}}
```

- Already

```
{"success":false,"message":"Shift already closed."}
```

45. Delete Shift

`URL: POST  /Shift/Delete`

Description: Delete by Id, Confirm=true.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	int	Yes	Shift id
Confirm	bool	Yes	Must be true

Request example:

```
{
```

==========

```
  "Id": 4,
  "Confirm": true
}
```

Response examples:

- OK

```
{"success":true,"message":"Shift deleted successfully"}
```

9. Fee

Calculate fees via DB function, query due/paid amounts, manage fee rules.

46. Calculate Fee

`URL: POST / GET  /Fee/Calculate`

Description: Resolve entry time by code/ticket_id/plate and call dbo.Fn_GetParkFee; or pass Minutes + In_Time directly.

Request parameters (JSON Body):

Parameter	Type	Required	Description
code	string	No	Record code (preferred)
ticket_id	string	No	Ticket id
Plate	string	No	Plate (open record)
Minutes	int	No	Direct parked minutes
In_Time	datetime	No	Direct entry time

Request example:

```
{
```

==========

```
  "code": "P20260801130311346"
}
```

Response examples:

- OK

```
{"success":true,"data":{"selector":"Code","in_time":"2026-08-01
13:03:11","parked_minutes":180,"fee":900.00}}
```

- Error

```
{"success":false,"message":"Code Not Exist"}
```

47. Due Fee

URL: POST /Fee/Due

Description: Total fee - paid = due (min 0) for a Code.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code

Request example:

```
{
  "Code": "P..."
}
```

Response examples:

- OK

```
{"success":true,"data":{"code":"P...","in_time":"...","parked_minutes":180,"total_fee":900.00,"p
aid":500.00,"due_fee":400.00}}
```

==========

48. Paid Amount

URL: POST /Fee/Paid

Description: Sum of charges for a Code.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	Yes	Record code

Request example:

```
{
  "Code": "P..."
}
```

Response examples:

- OK

```
{"success":true,"data":{"code":"P...","paid":500.00}}
```

49. Get Fee Rules

URL: POST / GET /Fee/Rules/Get

Description: Returns Simple_Fee_Rules and the dbo.Fn_GetParkFee definition.

Response examples:

- OK

```
{"success":true,"data":{"simple_fee_rules":"[...]","fn_get_park_fee":"CREATE OR ALTER FUNCTION ..."}}
```

50. Set Simple Fee Rules

URL: POST /Fee/Rules/Set

==========

Description: Set minute-range rules, which overwrite Fn_GetParkFee and Simple_Fee_Rules. Ranges must not overlap.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Rules	array	Yes	Array of {Start,End,Fee} in minutes/amount

Request example:

```
{
  "Rules": [
    {"Start": 0, "End": 30, "Fee": 300},
    {"Start": 30, "End": 60, "Fee": 600}
  ]
}
```

Response examples:

- OK

```
{"success":true,"message":"Simple fee rules saved and applied."}
```

- Overlap

```
{"success":false,"message":"Time ranges must not overlap."}
```

51. Get Fee Function

URL: POST / GET /Fee/Function/Get

Description: Returns the full dbo.Fn_GetParkFee definition.

Response examples:

- OK

```
{"success":true,"data":{"fn_get_park_fee":"CREATE OR ALTER FUNCTION dbo.Fn_GetParkFee ..."}}
```

52. Set Fee Function

URL: POST /Fee/Function/Set

Description: Execute a full function script. Must define dbo.Fn_GetParkFee; CREATE FUNCTION is auto-rewritten to CREATE OR ALTER.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Script	string	Yes	Full T-SQL function script

Request example:

```
{
  "Script": "CREATE OR ALTER FUNCTION dbo.Fn_GetParkFee(@InTime DATETIME,@ParkMinutes INT,@Now DATETIME) RETURNS FLOAT AS BEGIN RETURN 300; END"
}
```

Response examples:

- OK

```
{"success":true,"message":"Function saved and applied."}
```

- Error

```
{"success":false,"message":"Script must define the function dbo.Fn_GetParkFee."}
```

53. Get Lost Ticket Fee

URL: POST /Fee/LostTicket/Get

Description: Returns Park_Info.Lost_Ticket_Fee.

Response examples:

- OK

```
{"success":true,"data":{"lost_ticket_fee":500.00}}
```

54. Set Lost Ticket Fee

URL: POST /Fee/LostTicket/Set

Description: Updates Park_Info.Lost_Ticket_Fee.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Lost_Ticket_Fee	number	Yes	Lost ticket fee

Request example:

```
{
  "Lost_Ticket_Fee": 500
}
```

Response examples:

- OK

```
{"success":true,"message":"Lost ticket fee saved."}
```

55. Charge Lost Ticket

URL: POST /Fee/LostTicket/Charge

Description: Write a lost-ticket charge (Is_Lost_Ticket=1) for a Code; fee defaults to Park_Info.Lost_Ticket_Fee.

Request parameters (JSON Body):

Parameter	Type	Required	Description

Code	string	Yes	Record code
Fee	number	No	Amount (default lost ticket fee)
Pay_Type	int	No	Pay type
User_id	string	No	Cashier

Request example:

```
{
  "Code": "P...",
  "Pay_Type": 0,
  "User_id": "admin"
}
```

Response examples:

- OK

```
{"success":true,"message":"Lost ticket charge recorded","data":{"code":"P...","fee":500.00}}
```

10. Operator (Park_Operator)

Manage cashier/operator accounts.

56. Query Operators

URL: POST / GET /Operator/Query

Description: Fuzzy Username/Name.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Username	string	No	Username (fuzzy)

Name	string	No	Name (fuzzy)

Request example:

```
{
  "Username": "admin"
}
```

Response examples:

- OK

```
{"success":true,"data":[{"username":"admin","password":"8210","name":"Administrator"}]}
```

57. Get Operator

URL: POST /Operator/Get

Description: Exact read by Username.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Username	string	Yes	Username

Request example:

```
{
  "Username": "admin"
}
```

Response examples:

- OK

```
{"success":true,"data":{"username":"admin","password":"8210","name":"Administrator"}}
```

==========

- Error

```
{"success":false,"message":"Operator not found"}
```

58. Operator Login

URL: POST /Operator/Login

Description: Validate username/password.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Username	string	Yes	Username
Password	string	Yes	Password

Request example:

```
{
  "Username": "admin",
  "Password": "8210"
}
```

Response examples:

- OK

```
{"success":true,"message":"Login successful","data":{"username":"admin","name":"Administrator"}}
```

- Error

```
{"success":false,"message":"Invalid username or password"}
```

59. Add Operator

URL: POST /Operator/Add

Description: Add an operator; username must be unique.

==========

Request parameters (JSON Body):

Parameter	Type	Required	Description
Username	string	Yes	Username
Password	string	Yes	Password
Name	string	No	Name

Request example:

```
{
  "Username": "cashier1",
  "Password": "123456",
  "Name": "Cashier One"
}
```

Response examples:

- OK

```
{"success":true,"message":"Operator added successfully"}
```

- Dup

```
{"success":false,"message":"This username already exists."}
```

60. Modify Operator

URL: POST /Operator/Modify

Description: Update password and/or name by Username.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Username	string	Yes	Username

Password	string	No	New password
Name	string	No	Name

Request example:

```
{
  "Username": "cashier1",
  "Name": "Cashier A"
}
```

Response examples:

- OK

```
{"success":true,"message":"Operator updated successfully"}
```

61. Delete Operator

`URL: POST  /Operator/Delete`

Description: Delete by Username.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Username	string	Yes	Username

Request example:

```
{
  "Username": "cashier1"
}
```

Response examples:

- OK

==========

{"success":true,"message":"Operator deleted successfully"}

11. Pictures (Park_Pictures)

Read/write ANPR snapshots; manage pending recognition and plate back-fill.

62. Get Picture (Base64)

URL: POST / GET /Picture/Get

Description: By Id returns Base64 content, length and plate.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	long	Yes	Picture id

Request example:

```
{
  "Id": 101
}
```

Response examples:

- OK

{"success":true,"data":{"id":101,"length":20345,"plate":"T367AYG","format":"jpg","base64":"/9j/4AAQ..."}}

- Error

{"success":false,"message":"Picture not found"}

63. Query Picture Meta

URL: POST / GET /Picture/Query

==========

Description: By Id or Plate returns metadata (no Base64). Plate_Null=true shows unrecognized only.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	long	No	Picture id
Plate	string	No	Plate (fuzzy)
Plate_Null	bool	No	true = plate IS NULL
Top	int	No	Max rows (default 50)

Request example:

```
{
  "Plate": "T367",
  "Top": 20
}
```

Response examples:

- OK

```
{"success":true,"data":[{"id":101,"plate":"T367AYG","size":20345}]}
```

64. Pending Pictures

`URL: POST  /Picture/Pending`

Description: Pictures with Plate IS NULL (awaiting ANPR).

Request parameters (JSON Body):

Parameter	Type	Required	Description
Top	int	No	Max rows (default 50)

==========

Request example:

```
{
  "Top": 100
}
```

Response examples:

- OK

```
{"success":true,"data":[{"id":110,"plate":null,"size":19500}]}
```

65. Insert Picture

URL: POST /Picture/Insert

Description: Insert a Base64 picture, optionally with a plate. Returns the new Id.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Base64	string	Yes	Base64 image content
Plate	string	No	Plate

Request example:

```
{
  "Base64": "/9j/4AAQ...",
  "Plate": "STN2171"
}
```

Response examples:

- OK

```
{"success":true,"message":"Picture inserted successfully","data":{"id":115}}
```

==========

66. Set Plate

URL: POST /Picture/SetPlate

Description: Set/correct the plate of a picture by Id.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	long	Yes	Picture id
Plate	string	Yes	Plate

Request example:

```
{
  "Id": 115,
  "Plate": "STN2171"
}
```

Response examples:

- OK

```
{"success":true,"message":"Plate updated successfully"}
```

67. Delete Picture

URL: POST /Picture/Delete

Description: Delete by Id, Confirm=true.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Id	long	Yes	Picture id

Confirm	bool	Yes	Must be true

Request example:

```
{
  "Id": 115,
  "Confirm": true
}
```

Response examples:

- OK

```
{"success":true,"message":"Picture deleted successfully"}
```

12. Report

Combined visit+charge report and various summaries.

68. Combined Report

URL: POST / GET /Report/Combined

Description: Park_Records FULL JOIN Park_ParkingChargingRecords; filters on plate/ticket/card/customer/device, in/out/charge time ranges, fee, Pay_Type, Shift_Id.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Code	string	No	Record code (fuzzy)
Plate	string	No	Plate (fuzzy)
Ticket_Id	string	No	Ticket id (fuzzy)
Card_Id	string	No	Card id (fuzzy)

Customer_Code	string	No	Customer code (fuzzy)
In_Device	string	No	Entry device (fuzzy)
Out_Device	string	No	Exit device (fuzzy)
Operator	string	No	Cashier (fuzzy)
Machine	string	No	Machine (fuzzy)
in_time_start	datetime	No	Entry time from
in_time_end	datetime	No	Entry time to
out_time_start	datetime	No	Exit time from
out_time_end	datetime	No	Exit time to
charge_dt_start	datetime	No	Charge time from
charge_dt_end	datetime	No	Charge time to
Fee_Amount_Min	number	No	Fee min (-1 = none)
Fee_Amount_Max	number	No	Fee max (-1 = none)
Pay_Type	int	No	Pay type (-1 = none)
Shift_Id	int	No	Shift id (-1 = none)

Request example:

```
{
  "Plate": "T367AYG"
}
```

Response examples:

- OK

==========

{"success":true,"data":[{"code":"P...","plate":"T367AYG","fee":20.00,"pay_type":0,"pay_type_name":"Cash","charge_dt":"2026-08-01 14:00:00","machine":"Casher0","operator":"admin","shift_id":3}]}

69. Daily Summary

URL: POST /Report/Daily

Description: Per-day count, fee and lost count.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Charge_Dt_start	datetime	No	Charge time from
Charge_Dt_end	datetime	No	Charge time to

Response examples:

- OK

{"success":true,"data":[{"day":"2026-08-01","count":300,"fee":12000.00,"lost_count":2}]}

70. Pay-type Summary

URL: POST /Report/PayTypeSummary

Description: Per-pay-type count and fee.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Charge_Dt_start	datetime	No	Charge time from
Charge_Dt_end	datetime	No	Charge time to

==========

Response examples:

- OK

```
{"success":true,"data":[{"pay_type":0,"pay_type_name":"Cash","count":200,"fee":8000.00}]}
```

71. Shift Summary

URL: POST /Report/ShiftSummary

Description: Per-shift count and fee.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Charge_Dt_start	datetime	No	Charge time from
Charge_Dt_end	datetime	No	Charge time to

Response examples:

- OK

```
{"success":true,"data":[{"shift_id":3,"count":12,"fee":560.00,"machine":"Casher0","operator":"admin"}]}
```

72. Operator Summary

URL: POST /Report/OperatorSummary

Description: Per-operator count and fee.

Request parameters (JSON Body):

Parameter	Type	Required	Description
Charge_Dt_start	datetime	No	Charge time from

Charge_Dt_end	datetime	No	Charge time to

Response examples:

- OK

```
{"success":true,"data":[{"operator":"admin","count":120,"fee":5000.00}]}
```

73. Device Summary

URL: POST /Report/DeviceSummary

Description: Entry count per entry device.

Request parameters (JSON Body):

Parameter	Type	Required	Description
in_time_start	datetime	No	Entry time from
in_time_end	datetime	No	Entry time to

Response examples:

- OK

```
{"success":true,"data":[{"in_device":"ENTRY#1","count":150}]}
```

13. ANPR

Server-side license-plate recognition status and recent results.

74. ANPR Records

URL: POST / GET /Anpr/Records

Description: Recent in-memory recognition records (with picture id).

==========

Request parameters (JSON Body):

Parameter	Type	Required	Description
Top	int	No	Max rows (default 100)

Request example:

```
{
  "Top": 50
}
```

Response examples:

- OK


```
{"success":true,"data":[{"time":"2026-08-01 13:03:11","sender":"Picture 101","result":"T367AYG","picture_id":101,"picture_path":""}]}
```


75. Clear ANPR Records

URL: POST /Anpr/Clear

Description: Clear the in-memory record list.

Response examples:

- OK


```
{"success":true,"message":"ANPR records cleared"}
```


76. ANPR Status

URL: POST / GET /Anpr/Status

Description: Whether the recognizer is running and the pending picture count.

Response examples:

- OK

==========

{"success":true,"data":{"anpr_running":true,"pending_pictures":0}}

Appendix A: Data Dictionary

Park_Records (visit records)

Field	Type	Description
Code	nvarchar	Unique visit code
Plate	nvarchar	Plate
Ticket_Id	nvarchar	Ticket id
Card_Id	nvarchar	Card id
Cust_Code	nvarchar	Customer code
In_Device	nvarchar	Entry device
In_Time	datetime	Entry time
In_Picture_Id	bigint	Entry picture id
Out_Device	nvarchar	Exit device
Out_Time	datetime	Exit time (null = in lot)
Out_Picture_Id	bigint	Exit picture id

Park_Customer (VIP)

Field	Type	Description
Code	nvarchar	10-digit customer code

Card_Id	nvarchar	Card id
Name	nvarchar	Name
Phone	nvarchar	Phone
Plate	nvarchar	Plate
Reg_Dt	datetime	Registration date
Expire_Dt	datetime	Expiry date

Park_ParkingChargingRecords

Field	Type	Description
Code	nvarchar	Visit code
Fee	decimal	Fee amount
Pay_Type	int	Pay type
Charge_Dt	datetime	Charge time
Machine	nvarchar	Machine
Operator	nvarchar	Cashier
Shift_Id	int	Shift id (needs Update_Shift.sql)
Is_Lost_Ticket	bit	Lost ticket (needs Update_Shift.sql)

Park_VIPChargingRecords

Field	Type	Description

Id	bigint	PK
Cust_Code	nvarchar	Customer code
Card_Id	nvarchar	Card id
Name	nvarchar	Name
Operation	nvarchar	Add / Extend
Period_Start	datetime	Validity from
Period_End	datetime	Validity to
Days	int	Days
Rate	decimal	Daily rate
Fee	decimal	Amount
Pay_Type	int	Pay type
Charge_Dt	datetime	Charge time
Machine	nvarchar	Machine
Operator	nvarchar	Cashier

Park_Shifts (needs Update_Shift.sql)

Field	Type	Description
Id	int	PK
Machine	nvarchar	Machine
Operator	nvarchar	Cashier

Start_Dt	datetime	Start
End_Dt	datetime	End
Status	int	0=Open 1=Closed
Total_Fee	decimal	Total fee
Charge_Count	int	Charge count
Lost_Count	int	Lost count
Lost_Amount	decimal	Lost amount

Park_Info

Field	Type	Description
Total_Spaces	int	Total spaces
Limited_Exit_Time	int	Exit time limit (min)
Lost_Ticket_Fee	decimal	Lost ticket fee
Simple_Fee_Rules	nvarchar	Simple fee rules JSON

Park_Operator

Field	Type	Description
Username	nvarchar	Username
Password	nvarchar	Password
Name	nvarchar	Name

Park_Pictures

Field	Type	Description
Id	bigint	PK
Picture	varbinary	Image bytes
Plate	nvarchar	Plate (NULL = pending)

Park_Objects

Field	Type	Description
Object	nvarchar	Enum name e.g. Pay_Type
Value	int	Enum value
Name	nvarchar	Display name

Appendix B: Fee Function & Rules

Fees are computed by the DB scalar function dbo.Fn_GetParkFee:

```
dbo.Fn_GetParkFee(@InTime DATETIME, @ParkMinutes INT, @Now DATETIME) RETURNS FLOAT
```

@InTime=entry time, @ParkMinutes=parked minutes (>=1), @Now=current time; returns FLOAT amount.

Simple rules (/Fee/Rules/Set) auto-generate this function from minute ranges; complex rules (/Fee/Function/Set) write a full script.

==========

Appendix C: Common Errors

Message	Meaning
No such API endpoint: ...	Unknown path or method (HTTP 404).
Database error: ...	Database connection/execution failed.
Server error: ...	Unexpected server exception.
No 'Code' Selector Provided	Required parameter Code missing.
Code Not Exist	The visit record does not exist.
Customer not found	VIP customer does not exist.
Already Has A Same Plate / Card_id	Duplicate plate or card id.
Out of validity period	VIP expired.
Cannot Enter Repeatedly.	Open record exists; cannot enter again.
Fee Value Lacked Or Incorrect Or Json Body Incorrect	Fee missing or malformed.
Pay_Type Value Lacked Or Incorrect	Pay type missing or invalid.
User_id Value Lacked	Cashier missing.
No Selector Provided or Format Error	No query condition provided or bad JSON.