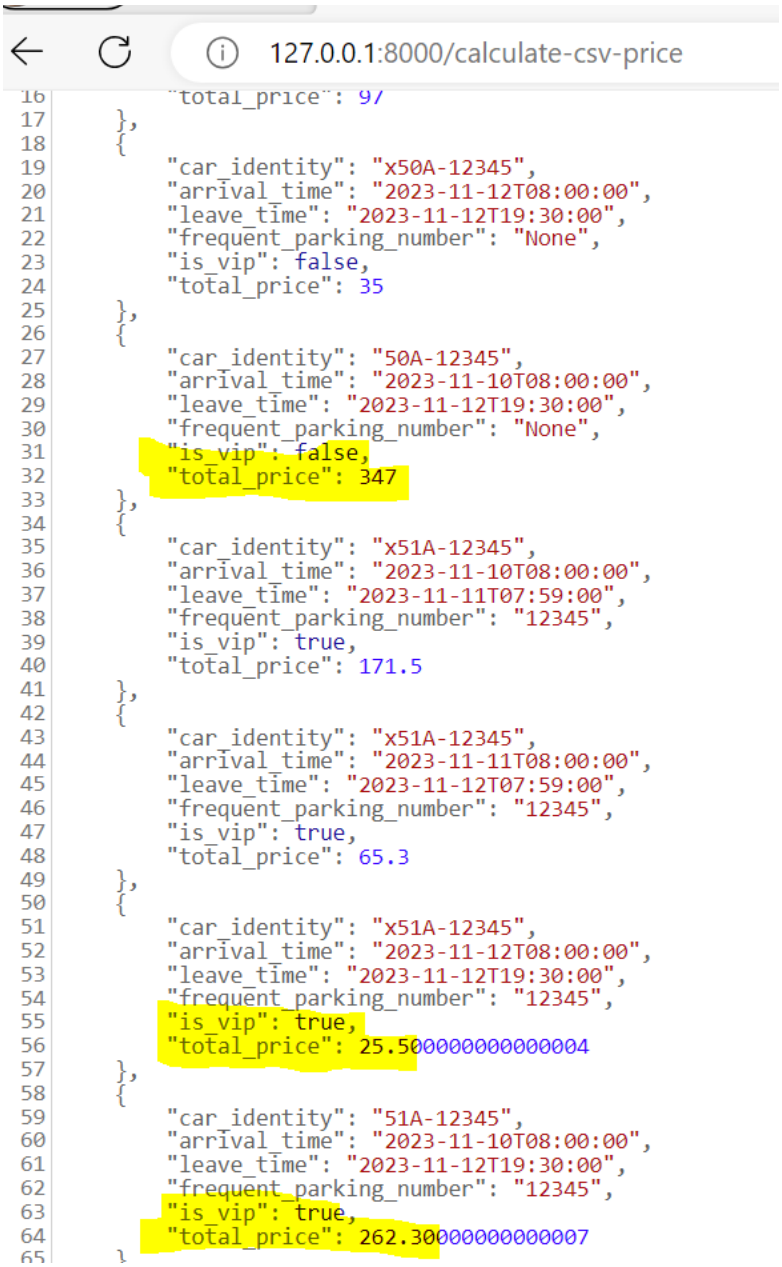
Contents

Calculate price by reading csv

Monday, December 4, 2023
2:32 PM

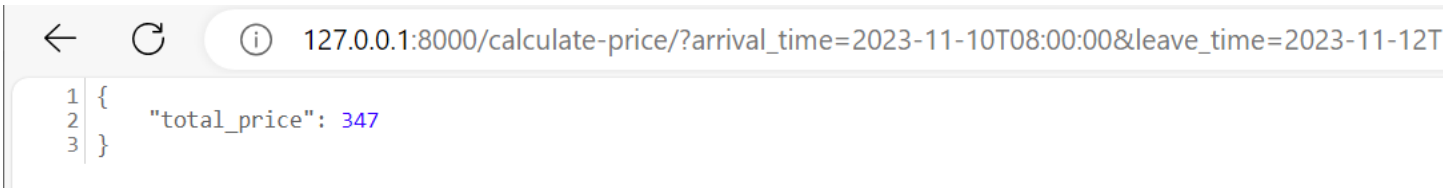
127.0.0.1:8000/calculate-csv-price



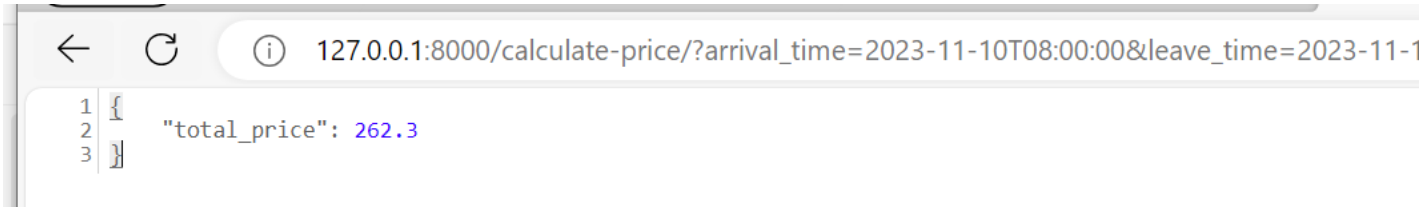
Calculate price directly by api

Monday, December 4, 2023
2:33 PM

http://127.0.0.1:8000/calculate-price/?arrival_time=2023-11-10T08:00:00&leave_time=2023-11-12T19:30:00&is_vip=false



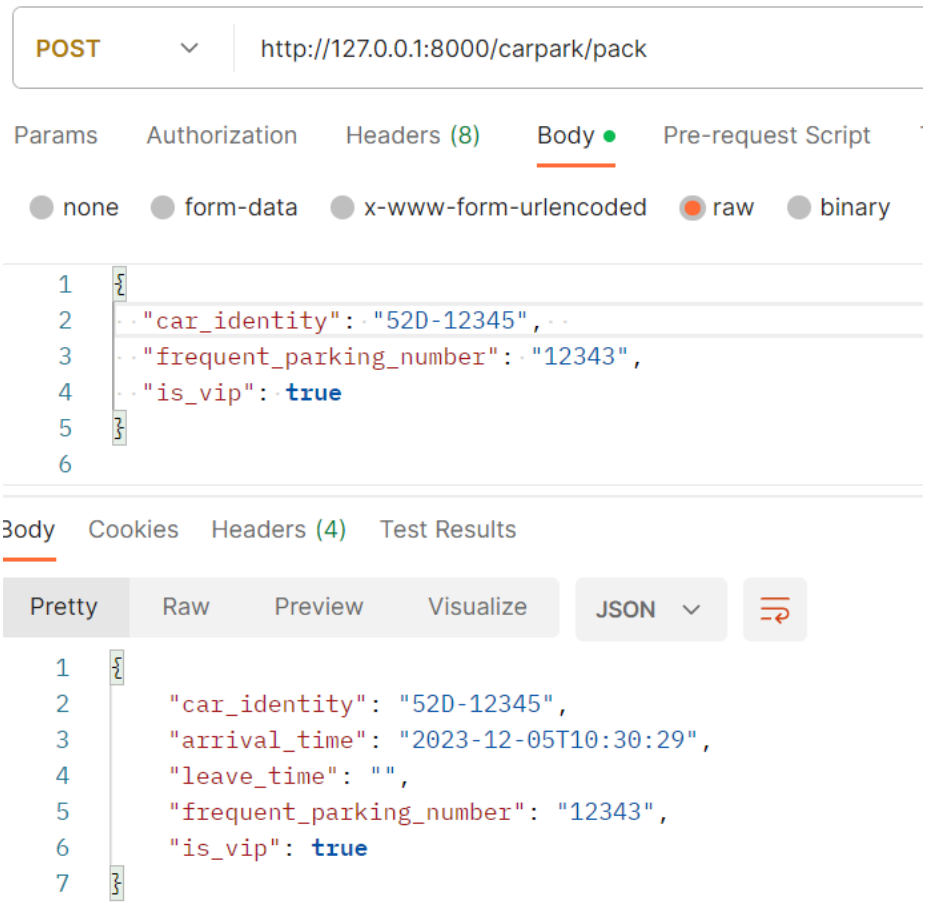
127.0.0.1:8000/calculate-price/?arrival_time=2023-11-10T08:00:00&leave_time=2023-11-12T19:30:00&is_vip=true

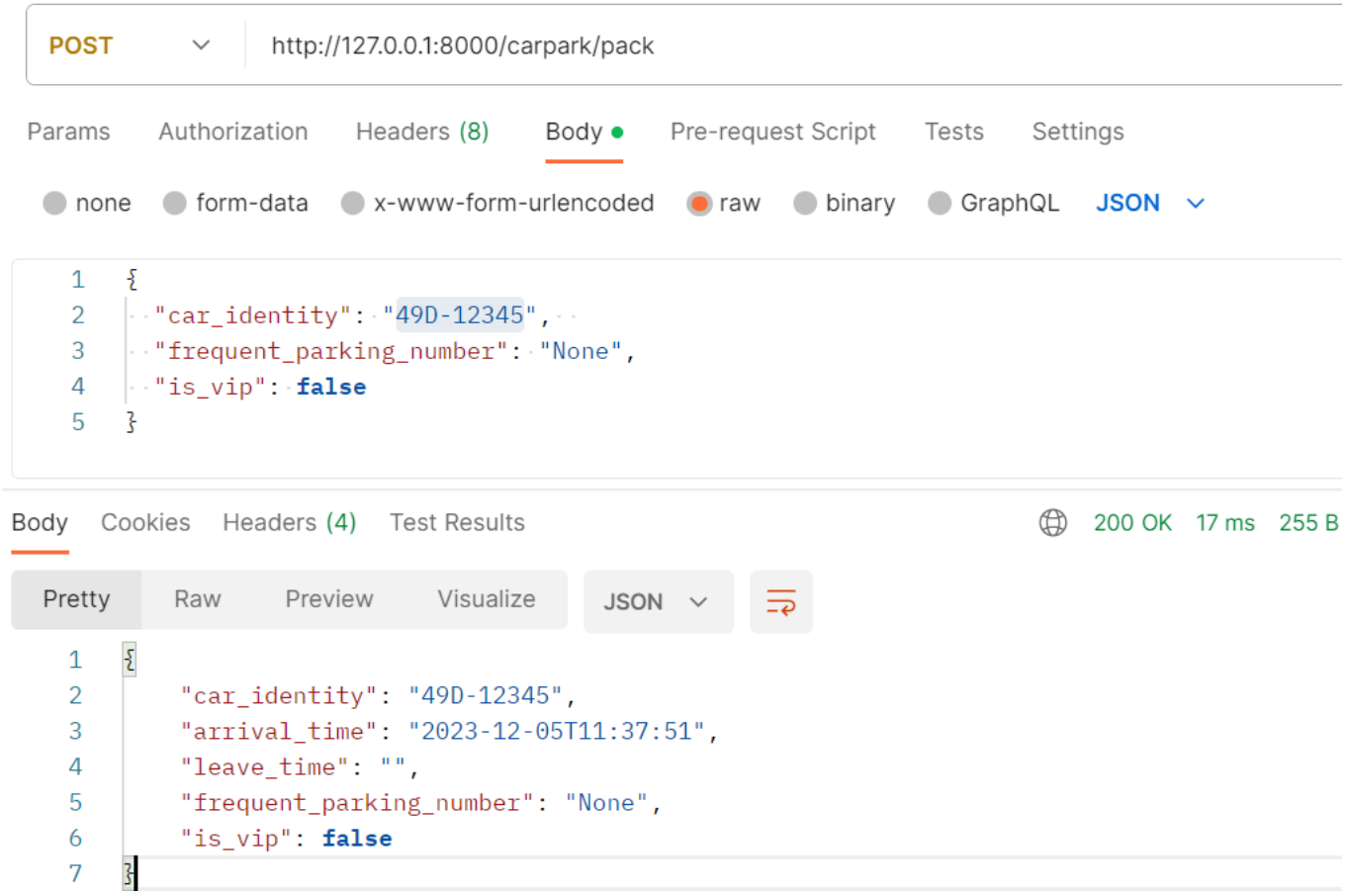



```python
@app.get("/calculate-price/")
async def calculate_price_endpoint(
    arrival_time: str = Query(..., description="Arrival time in ISO format"),
    leave_time: str = Query(..., description="Leave time in ISO format"),
    is_vip: Optional[bool] = Query(False, description="Whether the owner is a VIP member"),
):
    try:
        packing_system = ParkingSystem()
        total_price = packing_system.calculate_parking_price({"arrival_time":arrival_time,
        return {"total_price": total_price}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

pack

Monday, December 4, 2023
8:26 PM


POST
http://127.0.0.1:8000/carpark/pack
Params
Authorization
Headers (8)
Body
Pre-request Script
none
form-data
x-www-form-urlencoded
raw
binary
{
"car_identity": "52D-12345",
"frequent_parking_number": "12343",
"is_vip": true
}
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"car_identity": "52D-12345",
"arrival_time": "2023-12-05T10:30:29",
"leave_time": "",
"frequent_parking_number": "12343",
"is_vip": true
}



Pack again will cause validation error.

Python Ex1 / **/carpark/pack valid**

Save

POST http://127.0.0.1:8000/carpark/pack

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "car_identity": "52D-12345",
  "frequent_parking_number": "12343",
  "is_vip": true
}
```

Body Cookies Headers (4) Test Results 400 Bad Request 11 ms 186 B Sa

Pretty Raw Preview Visualize JSON

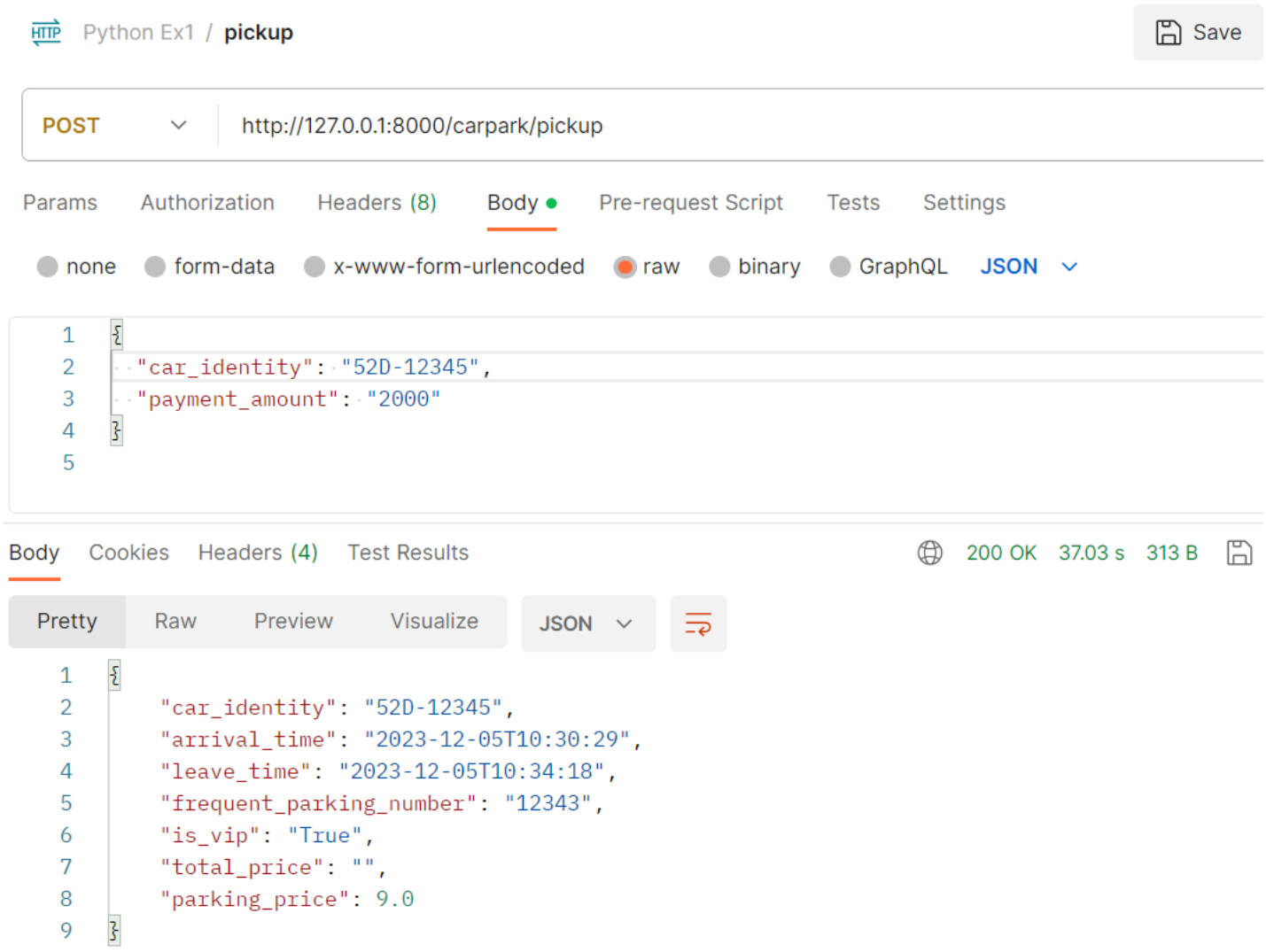
```
{
    "detail": "Car already parked and has not left yet"
}
```

```python
@app.post("/carpark/pack")
async def pack_car(carpark: CarParkInModel):
    try:
        packing_system = ParkingSystem()
        output_carpark = packing_system.pack(carpark)
        return output_carpark
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

pickup

Monday, December 4, 2023
8:34 PM

Pickup will find the car with empty leave_time, then set leave time and total price



Pick up again, no car with empty leav_time is found, raise error

POST http://127.0.0.1:8000/carpark/pickup

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

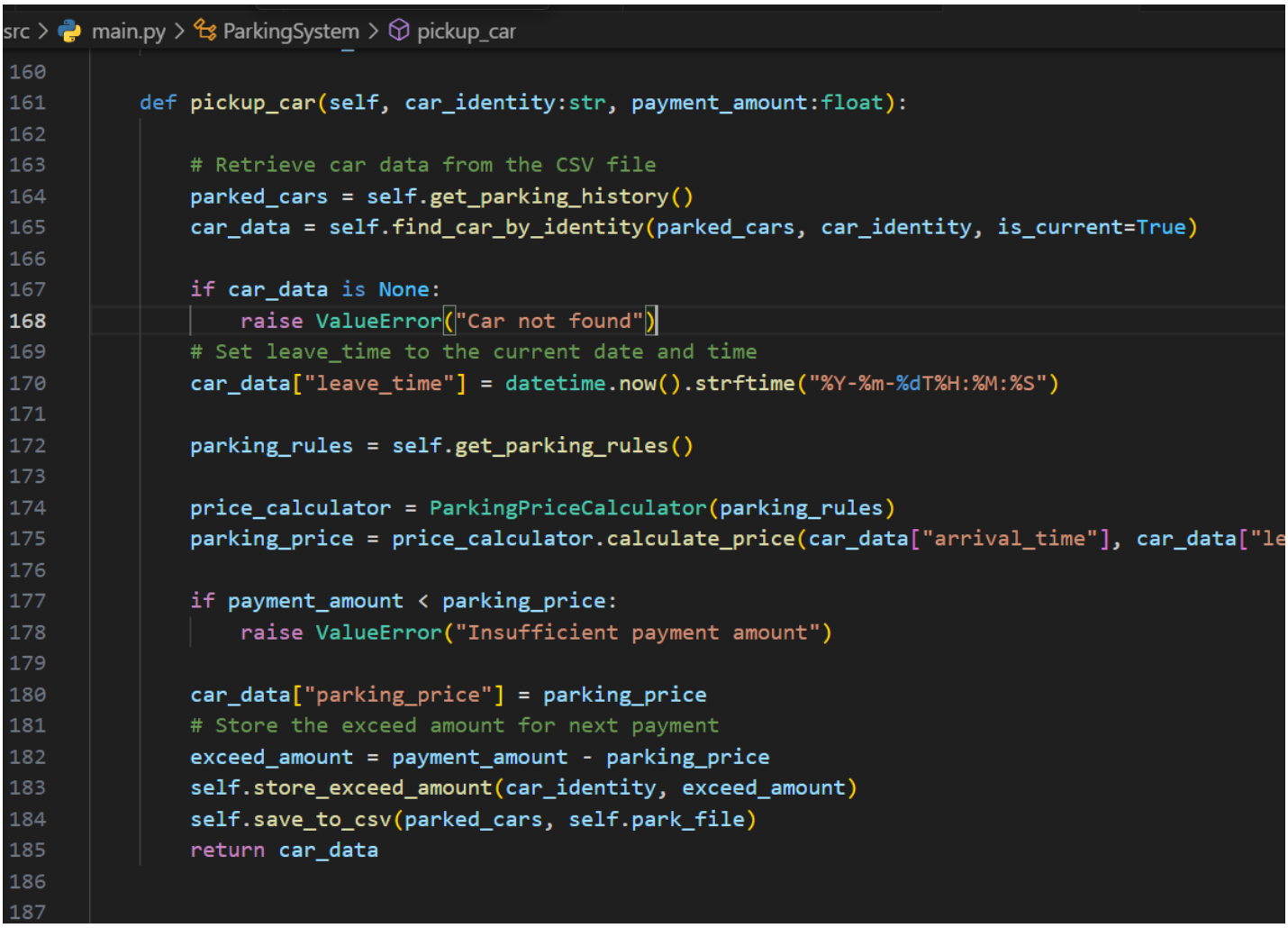
```
{
  "car_identity": "57D-12345",
  "payment_amount": "2000"
}
```

Body Cookies Headers (4) Test Results 400 Bad Request 20 ms 160 B

Pretty Raw Preview Visualize JSON

```
{
    "detail": "Car not found"
}
```

```python
@app.post("/carpark/pickup")
async def pickup_car(pickup_request: PickupRequest):
    try:
        packing_system = ParkingSystem()
        output_carpark = packing_system.pickup_car(pickup_request.car_identity, pickup_reque
        return output_carpark
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```


src > main.py > ParkingSystem > pickup_car
160
161 def pickup_car(self, car_identity:str, payment_amount:float):
162
163 # Retrieve car data from the CSV file
164 parked_cars = self.get_parking_history()
165 car_data = self.find_car_by_identity(parked_cars, car_identity, is_current=True)
166
167 if car_data is None:
168 raise ValueError("Car not found")
169 # Set leave_time to the current date and time
170 car_data["leave_time"] = datetime.now().strftime("%Y-%m-%dT%H:%M:%S")
171
172 parking_rules = self.get_parking_rules()
173
174 price_calculator = ParkingPriceCalculator(parking_rules)
175 parking_price = price_calculator.calculate_price(car_data["arrival_time"], car_data["le
176
177 if payment_amount < parking_price:
178 raise ValueError("Insufficient payment amount")
179
180 car_data["parking_price"] = parking_price
181 # Store the exceed amount for next payment
182 exceed_amount = payment_amount - parking_price
183 self.store_exceed_amount(car_identity, exceed_amount)
184 self.save_to_csv(parked_cars, self.park_file)
185 return car_data
186
187

```python
class ParkingPriceCalculator:
    def __init__(self, rules):
        self.rules = rules

    def calculate_price(self, arrival_time_str, leave_time_str, is_vip):
        arrival_time = datetime.strptime(arrival_time_str, "%Y-%m-%dT%H:%M:%S")
        leave_time = datetime.strptime(leave_time_str, "%Y-%m-%dT%H:%M:%S")
       # print(arrival_time)
       # print(leave_time)
        total_price = 0
        current_time = arrival_time

        while current_time < leave_time:
           # print("total_price before add",total_price)
            price = self.get_hourly_price(current_time, arrival_time, is_vip)
            total_price += price
            current_time += timedelta(hours=1)
           # print("-----current_time:",current_time, "total_price after add:", total_pric
        return round(total_price,2)

    def get_hourly_price(self, current_time, arrival_time, is_vip):
       # print("get_hourly_price", current_time, is_vip);
        weekday = current_time.strftime("%A")
```

src > pack-history.csv

```
car_identity,arrival_time,leave_time,frequent_parking_number,is_vip,total_price,parking
59C-12345,2023-12-04T10:00:00,2023-01-01T15:00:00,12343,TRUE,0,
52C-12345,2023-01-01T10:00:00,,12343,True,,
52D-12345,2023-12-05T10:30:29,2023-12-05T10:34:18,12343,True,,9.0

```

rc > exceed_amounts.csv

```
car_identity,exceed_amount
52D-12345,1991.0
```

History

Tuesday, December 5, 2023
12:00 PM

GET http://127.0.0.1:8000/carpark/history/?car_identity=52D-12345

Params • Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (4) Test Results 200 OK

Pretty Raw Preview Visualize JSON


```
[
    {
        "car_identity": "52D-12345",
        "arrival_time": "2023-12-05T10:30:29",
        "leave_time": "2023-12-05T10:34:18",
        "frequent_parking_number": "12343",
        "is_vip": "True",
        "total_price": "",
        "parking_price": "9.0"
    },
    {
        "car_identity": "52D-12345",
        "arrival_time": "2023-12-05T11:35:37",
        "leave_time": "",
        "frequent_parking_number": "12343",
        "is_vip": "True",
        "total_price": "",
        "parking_price": ""
    }
```


HTTP Python Ex1 / **http://127.0.0.1:8000/carpark/history/?car_identity=62D-12345**

GET http://127.0.0.1:8000/carpark/history/?car_identity=62D-12345

Params • Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value	Des
☑	car_identity	62D-12345	
	Key	Value	Des

Body Cookies Headers (4) Test Results 200 O

Pretty Raw Preview Visualize JSON


```
[]
```

```python
@app.get("/carpark/history")
async def car_history_endpoint(car_identity: str = Query(..., description="car identity")):
    try:
        packing_system = ParkingSystem()
        output_carpark = packing_system.car_history(car_identity)
        return output_carpark
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

main.py > ParkingSystem > car_history

```python
    def car_history(self, car_identity: str):
        parking_history = self.get_parking_history()
        ret = []
        if parking_history is not None:
            for row in parking_history:
                if row['car_identity'] == car_identity :
                    ret.append(row)
        return ret
```

PackingSystem

Tuesday, December 5, 2023
12:04 PM

```python
class ParkingSystem:
    def __init__(self):
        # Assuming you have a list to store the car data
        self.rule_file = "D:\\le\\nash\\python-ass\\ass1\\src\\ParkingRule.csv"
        self.park_file = "D:\\le\\nash\\python-ass\\ass1\\src\\pack-history.csv"
        self.exceed_amounts_file= "D:\\le\\nash\\python-ass\\ass1\\src\\exceed_amounts.csv"
    def get_parking_rules(self):
        data_loader = DataLoader()
        rules_data = data_loader.load_data(self.rule_file)
        return rules_data

    def get_parking_history(self):
        data_loader = DataLoader()
        parking_history = data_loader.load_data(self.park_file)
        self.parked_cars = parking_history
        return parking_history

    def find_car_by_identity(self, parking_history, car_identity: str, is_current: bool = True):
        if parking_history is not None:
            for row in parking_history:
                if row['car_identity'] == car_identity and (not is_current or not row['leave_time']):
                    return row
        return None

    def pack(self, carpark: CarParkInModel):
```

```python
def pack(self, carpark: CarParkInModel):
    parked_cars = self.get_parking_history()
    existing_car = self.find_car_by_identity(parked_cars, carpark.car_identity, is_current=True)

    if existing_car:
        raise ValueError("Car already parked and has not left yet")

    arrival_time = datetime.now().strftime("%Y-%m-%dT%H:%M:%S")
    saved_item = {'car_identity': carpark.car_identity, 'arrival_time': arrival_time, 'leave_time
    parked_cars.append(saved_item)
    self.save_to_csv(parked_cars, self.park_file)
    return saved_item
```

```python
def pickup_car(self, car_identity:str, payment_amount:float):
    # Retrieve car data from the CSV file
    parked_cars = self.get_parking_history()
    car_data = self.find_car_by_identity(parked_cars, car_identity, is_current=True)

    if car_data is None:
        raise ValueError("Car not found")
    # Set leave_time to the current date and time
    car_data["leave_time"] = datetime.now().strftime("%Y-%m-%dT%H:%M:%S")

    parking_rules = self.get_parking_rules()

    price_calculator = ParkingPriceCalculator(parking_rules)
    parking_price = price_calculator.calculate_price(car_data["arrival_time"], car_data

    if payment_amount < parking_price:
        raise ValueError("Insufficient payment amount")

    car_data["parking_price"] = parking_price
    # Store the exceed amount for next payment
    exceed_amount = payment_amount - parking_price
    self.store_exceed_amount(car_identity, exceed_amount)
    self.save_to_csv(parked_cars, self.park_file)
    return car_data
```

```python
def calculate_parking_price(self, car_data: dict) -> float:
    data_loader = DataLoader()
    parking_rules = data_loader.load_data(self.rule_file)
    price_calculator = ParkingPriceCalculator(parking_rules)
    parking_price = price_calculator.calculate_price(car_data["arrival_
    return parking_price


def save_to_csv(self, data: List[dict], file_path: str):
    if data:
        df = pd.DataFrame(data)
        df.to_csv(file_path, index=False)
```

```python
def car_history(self, car_identity: str):
    parking_history = self.get_parking_history()
    ret = []
    if parking_history is not None:
        for row in parking_history:
            if row['car_identity'] == car_identity :
                ret.append(row)
    return ret
```