Carpark Availability Automation System Documentation

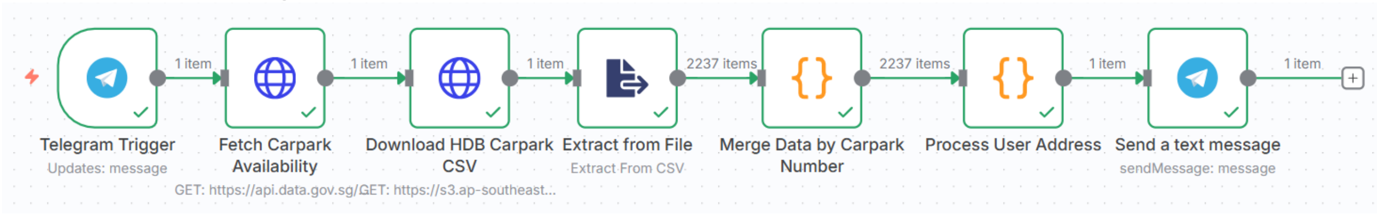
Project Overview

This n8n workflow automates the process of finding available carpark lots across different areas in Singapore. The system integrates real-time carpark availability data with comprehensive carpark information, providing users with instant access to parking availability through Telegram.

Workflow Architecture

Node Sequence:

1. **Telegram Trigger** - Listen for user messages containing area names
2. **Fetch Carpark Availability** - Retrieve real-time data from `api.data.gov.sg/v1/transport/carpark-availability`
3. **Download HDB Carpark CSV** - Fetch static carpark information dataset
4. **Extract from File** - Process the CSV data
5. **Merge Data by Carpark Number** - Combine real-time availability with static carpark details
6. **Process User Address** - Filter data based on user's area query
7. **Send Text Message** - Return formatted results via Telegram



Data Integration

Real-time Data Source:

- **API Endpoint**: `https://api.data.gov.sg/v1/transport/carpark-availability`
- **Data Provided**: Live availability of carpark lots (updated frequently)
- **Key Information**: Number of available lots, total lots, lot types (Cars, Motorcycles, Heavy Vehicles)

Static Data Source:

- **Dataset**: HDB Carpark Information from https://data.gov.sg/datasets/d_23f946fa557947f93a8043bbef41dd09/view
- **Enrichment Data**: Physical addresses, coordinates, carpark types, parking system details
- **Key Benefit**: Adds contextual information (locations, addresses) to the raw availability data

User Experience

1. User sends a Telegram message with an area name (e.g., "Bukit Batok West Avenue 6")
2. System processes the query and searches through 2,237 carparks
3. Return filtered results showing:
   - Carpark addresses in the specified area
   - Carpark identification numbers
   - Real-time availability of car lots (Type C)

Benefits of Using n8n

1. No-Code/Low-Code Automation

- Visual workflow builder enables rapid development without extensive programming knowledge
- Easy modification and maintenance of the automation process

2. Seamless Integration Capabilities

- Native connectors for Telegram, HTTP requests, and file processing
- Ability to combine multiple data sources into a unified workflow

3. Scalability and Reliability

- Handles large datasets (2,237+ carparks) efficiently
- Built-in error handling and retry mechanisms for API calls

4. Real-time Processing

- Instant response to user queries with live data
- Automated data refresh from government APIs

5. Cost-Effective Solution

- Reduce manual effort in checking multiple parking apps/websites
- Leverage free public data sources maintained by Singapore government

6. Customizable Output

- Flexible response formatting for optimal user experience
- Ability to extend functionality with additional data processing

Technical Advantages

Data Enrichment

The system demonstrates powerful data integration by merging:

- **Dynamic data** (real-time availability)
- **Static data** (location information)
- **User input** (area search queries)

Error Handling

- Graceful handling of missing data fields
- User-friendly error messages
- Fallback mechanisms for unavailable data

Conclusion

This n8n automation project successfully addresses the practical need for centralized carpark information in Singapore. By leveraging government open data APIs and Telegram's messaging platform, the system provides residents with a convenient, real-time solution for parking availability checks.

The implementation demonstrates how n8n serves as a powerful integration platform that can:

- Connect diverse data sources seamlessly
- Process complex data merging operations
- Deliver practical solutions through popular communication channels
- Reduce development time through visual workflow automation

This project not only solves an immediate user need but also provides a template for other similar public service automation projects that can enhance urban living through smart use of available data resources.

Code Snippet Explanations

Merge Data by Carpark Number

```
# Get the API data from the 'Fetch Carpark Availability' node
api_data = _("Fetch Carpark Availability").first().json["items"][0].carpark_data
```

- `_("Fetch Carpark Availability")`: Get the output from the node named "Fetch Carpark Availability"
- `.first()`: Take the first item from that node's output
- `.json`: Access the JSON data of that item
- `["items"][0]`: Access the first element in the "items" array
- `["carpark_data"]`: Extract the "carpark_data" field from that element
- Get the carpark availability data from the API response

```
csv_data = _input.all()
```

- `_input.all()`: Get all input data from the connected node (presumably a CSV node)
- Retrieve all the CSV data that will be merged with the API data

```
# Extract JSON from all CSV input items
all_csv_data = []
for i in range(len(csv_data)):
    all_csv_data.append(csv_data[i].json)
```

- Create an empty list to store CSV data
- Loop through all items in the CSV data
- For each item, extract the JSON content and add it to the list
- Convert the CSV input into a format that's easier to work with

```
# Create mapping from carpark number to availability
availability_map = {}
for carpark in api_data:
    availability_map[carpark['carpark_number']] = carpark['carpark_info']
```

- Create an empty dictionary to map carpark numbers to their availability info
- Loop through each carpark in the API data
- For each carpark, use the carpark number as the key and the availability info as the value
- Create a quick lookup table for availability data

```
# Merge the data
merged_data = []
for carpark in all_csv_data:
    carpark_number = carpark['car_park_no']
    # Create a new dictionary with all properties from carpark plus availability
    merged_item = {**carpark, 'availability': availability_map.get(carpark_number, [])}
    merged_data.append(merged_item)
```

- Create an empty list for the merged data
- Loop through each carpark in the CSV data
- Extract the carpark number from the CSV data
- Create a new dictionary that combines all the CSV data with the availability information from the API
- `**` (double asterisk) in front of `carpark is` a dictionary unpacking operator. This "unpacks" all the key-value pairs from the `carpark` dictionary
- That single line is equivalent to writing:
  - `merged_item = carpark.copy()  # Create a copy of the original dictionary`
  - `merged_item['availability'] = availability_map.get(carpark_number, [])  # Add new key`
- Uses `availability_map.get(carpark_number, [])` to get the availability data, defaulting to an empty list if not found
- Add the merged item to the result list

```
# Return each item as a separate item with json key
output = [{'json': item} for item in merged_data]
```

- Create the final output in the format expected by n8n
- Each item in the merged data is wrapped in a dictionary with a 'json' key
- This is the standard format for passing data between nodes in n8n

```
return output
```

- Return the formatted output so it can be passed to the next node in the workflow

This code essentially:

1. Fetch real-time availability data from an API
2. Get static carpark information from a CSV file
3. Merge them by matching carpark numbers
4. Return the combined data for use in subsequent nodes

The result is a dataset that contains both the static information from the CSV and the dynamic availability information from the API.

Process User Address

```
# Get all incoming items
items = _input.all()
```

- `_input.all()`: Retrieve all input data from the previous node in the n8n workflow

```
# Grab Telegram message from the Telegram Trigger node
telegram_node = _("Telegram Trigger")
```

- `_("Telegram Trigger")`: Access the output from the node named "Telegram Trigger"

```
telegram = telegram_node.first().json if telegram_node else None
```

- `.first().json`: Get the first item from the Telegram node and extracts its JSON data
- `if telegram_node else None`: Handle the case where the Telegram node might not exist

```
if not telegram or not telegram.get('message') or not telegram['message'].get('text'):
    raise Exception("Telegram message text not found.")
```

- Check if the Telegram message exists and contains text
- Raise an exception if any required part is missing

```
original_query = telegram['message']['text']
query = original_query.strip()
q_norm = query.lower()
```

- Extract the original message text
- Remove any leading/trailing whitespace
- Convert to lowercase for case-insensitive matching

```
# Consider only items that look like carpark records (have an address string)
carparks = [i for i in items if i.json and isinstance(i.json.get('address'), str)]
```

- Use a list comprehension to filter items
- Keep only items that have JSON data with an address field that is a string

```
# Match by case-insensitive substring
matched = [i for i in carparks if q_norm in i.json['address'].lower()]
```

- Another list comprehension to find carparks whose address contains the normalized query
- Use Python's `in` operator for substring matching

```
# Build response text
if len(matched) == 0:
    response = f'No carparks found for "{query}". Please try a different address or area name.'
```

- Handle the case where no matching carparks are found
- Use an f-string for string interpolation

```
else:
    list_items = []
    for idx, cp in enumerate(matched):
```

- Initialize an empty list for the formatted carpark information
- Use `enumerate()` to get both the index and item from the matched list

```
        j = cp.json
```

- Store the JSON data of the current carpark in a variable for easier access

```
        # Extract car lot availability (type 'C')
        car_lots_info = "No car lot availability data"
```

- Set a default value for car lot information

```
        if j.get('availability') and len(j['availability']) > 0:
```

- Check if availability data exists and is not empty

```
            car_availability = next((avail for avail in j['availability'] if avail.get('lot_type') == "C"), None)
```

- Use a generator expression with next() to find the first availability entry with lot_type "C"

```
        if car_availability:
                car_lots_info = f"Car Lots: {car_availability.get('lots_available', 'N/A')}/{car_availability.get('total_lots', 'N/A')} available"
```

- If car availability data is found, formats it into a string
- Use .get() with default values to handle missing data

```
        list_items.append(f"{idx + 1}. {j['address']}\n   Carpark Number: {j.get('car_park_no', 'N/A')}\n   {car_lots_info}")
```

- Format the carpark information into a string and adds it to the list
- Use f-string for string interpolation

```
    list_text = "\n\n".join(list_items)
```

- Join all the formatted carpark strings with double newlines

```
    response = f'Found {len(matched)} carparks for "{query}":\n\n{list_text}\n'
```

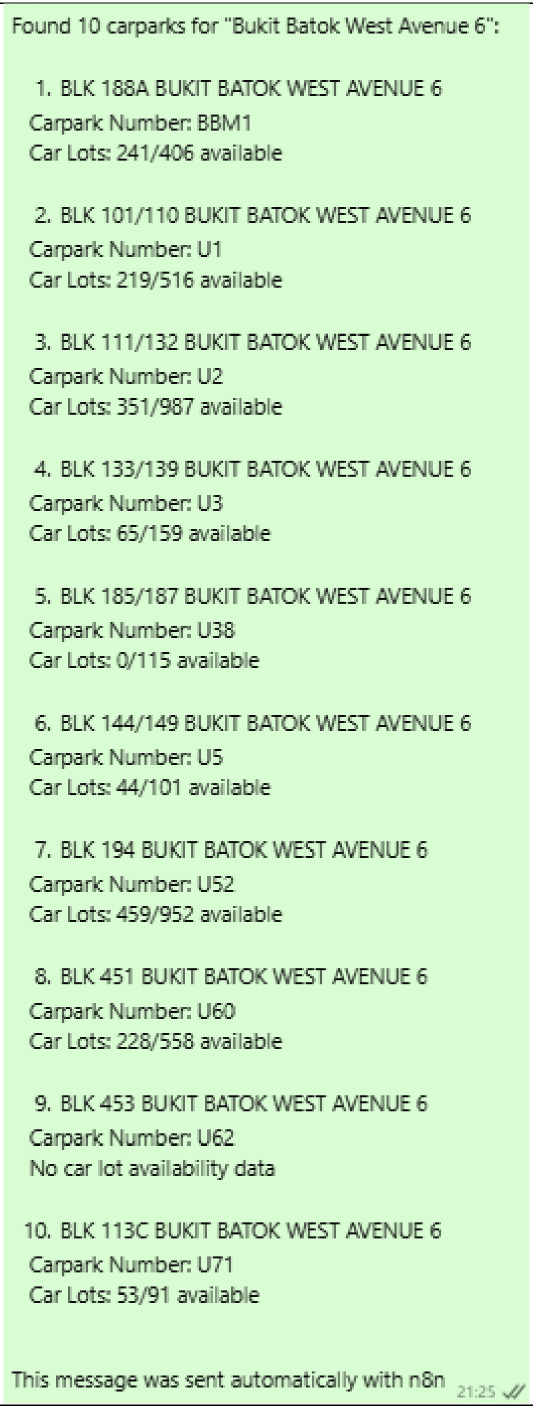
- Create the final response string with a header and the list of carparks

```
# Return response to Telegram
output = [{
    'json': {
        'chatId': telegram['message']['chat']['id'],
        'response': response
    }
}]
```

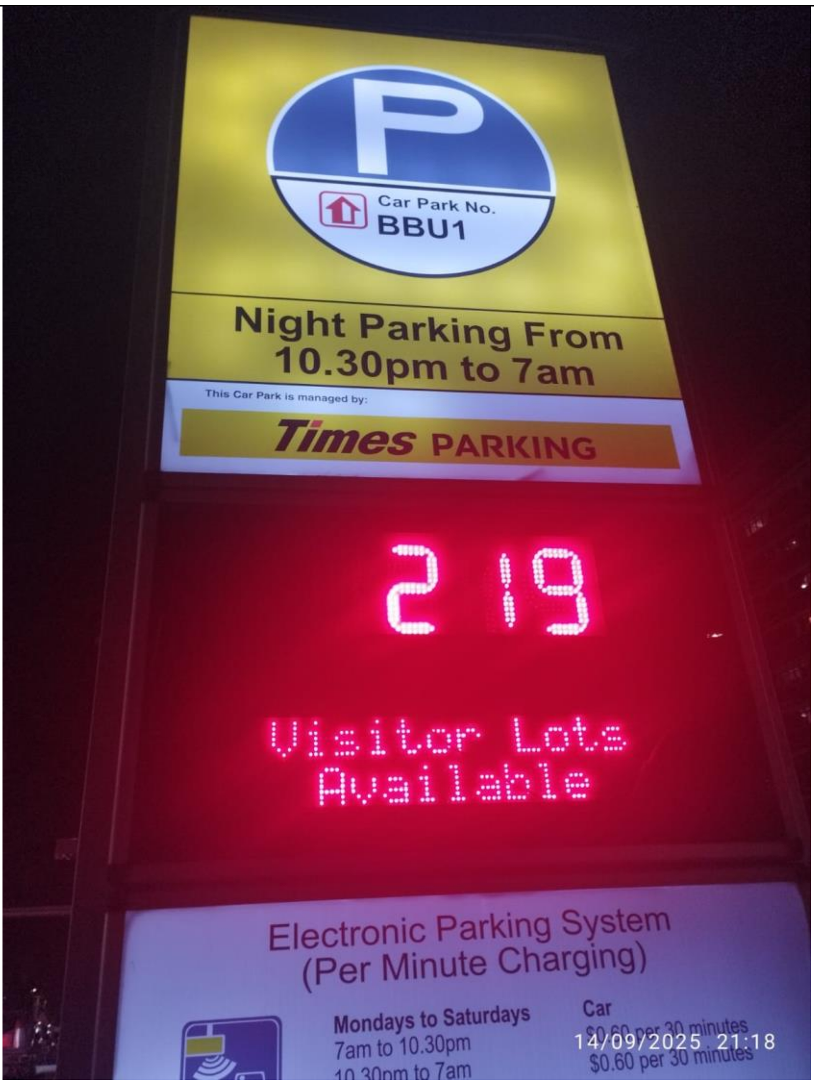
- Format the output in the structure expected by n8n
- Include the chat ID from the original Telegram message and the constructed response

The primary purpose of the above code is to process a Telegram message containing a location query, search through available carpark data for matching addresses, and generate a formatted response with carpark availability information that can be sent back to the user through Telegram. Essentially, it creates a carpark search and information service within a Telegram chat interface.

Outputs



```
Found 10 carparks for "Bukit Batok West Avenue 6":

 1. BLK 188A BUKIT BATOK WEST AVENUE 6
Carpark Number: BBM1
Car Lots: 241/406 available

 2. BLK 101/110 BUKIT BATOK WEST AVENUE 6
Carpark Number: U1
Car Lots: 219/516 available

 3. BLK 111/132 BUKIT BATOK WEST AVENUE 6
Carpark Number: U2
Car Lots: 351/987 available

 4. BLK 133/139 BUKIT BATOK WEST AVENUE 6
Carpark Number: U3
Car Lots: 65/159 available

 5. BLK 185/187 BUKIT BATOK WEST AVENUE 6
Carpark Number: U38
Car Lots: 0/115 available

 6. BLK 144/149 BUKIT BATOK WEST AVENUE 6
Carpark Number: U5
Car Lots: 44/101 available

 7. BLK 194 BUKIT BATOK WEST AVENUE 6
Carpark Number: U52
Car Lots: 459/952 available

 8. BLK 451 BUKIT BATOK WEST AVENUE 6
Carpark Number: U60
Car Lots: 228/558 available

 9. BLK 453 BUKIT BATOK WEST AVENUE 6
Carpark Number: U62
No car lot availability data

10. BLK 113C BUKIT BATOK WEST AVENUE 6
Carpark Number: U71
Car Lots: 53/91 available


This message was sent automatically with n8n   21:25
```



The above shows the Carpark Number U1 available lots of 219 tallies exactly with what was output to Telegram.

Appendix – Python Code

Merge Data by Carpark Number

```
# Get the API data from the 'Fetch Carpark Availability' node
api_data = _("Fetch Carpark Availability").first().json["items"][0].carpark_data

csv_data = _input.all()

# Extract JSON from all CSV input items
all_csv_data = []
for i in range(len(csv_data)):
    all_csv_data.append(csv_data[i].json)

# Create mapping from carpark number to availability
availability_map = {}
for carpark in api_data:
    availability_map[carpark['carpark_number']] = carpark['carpark_info']
```

```
# Merge the data
merged_data = []
for carpark in all_csv_data:
   carpark_number = carpark['car_park_no']
   # Create a new dictionary with all properties from carpark plus availability
   merged_item = {**carpark, 'availability': availability_map.get(carpark_number, [])}
   merged_data.append(merged_item)

# Return each item as a separate item with json key
output = [{'json': item} for item in merged_data]
return output
```

Process User Address

```
# Get all incoming items
items = _input.all()

# Grab Telegram message from the Telegram Trigger node
telegram_node = _("Telegram Trigger")
telegram = telegram_node.first().json if telegram_node else None

if not telegram or not telegram.get('message') or not telegram['message'].get('text'):
   raise Exception("Telegram message text not found.")

original_query = telegram['message']['text']
query = original_query.strip()
q_norm = query.lower()

# Consider only items that look like carpark records (have an address string)
carparks = [i for i in items if i.json and isinstance(i.json.get('address'), str)]

# Match by case-insensitive substring
matched = [i for i in carparks if q_norm in i.json['address'].lower()]

# Build response text
if len(matched) == 0:
   response = f'No carparks found for "{query}". Please try a different address or area name.'
else:
   list_items = []
   for idx, cp in enumerate(matched):
      j = cp.json

      # Extract car lot availability (type 'C')
      car_lots_info = "No car lot availability data"
      if j.get('availability') and len(j['availability']) > 0:
         car_availability = next((avail for avail in j['availability'] if avail.get('lot_type') == "C"), None)
         if car_availability:
            car_lots_info = f"Car Lots: {car_availability.get('lots_available', 'N/A')}/{car_availability.get('total_lots', 'N/A')} available"

      list_items.append(f"{idx + 1}. {j['address']}\n   Carpark Number: {j.get('car_park_no', 'N/A')}\n   {car_lots_info}")

   list_text = "\n\n".join(list_items)
   response = f'Found {len(matched)} carparks for "{query}":\n\n{list_text}\n'

# Return response to Telegram
output = [{
   'json': {
      'chatId': telegram['message']['chat']['id'],
      'response': response
   }
}]
return output
```