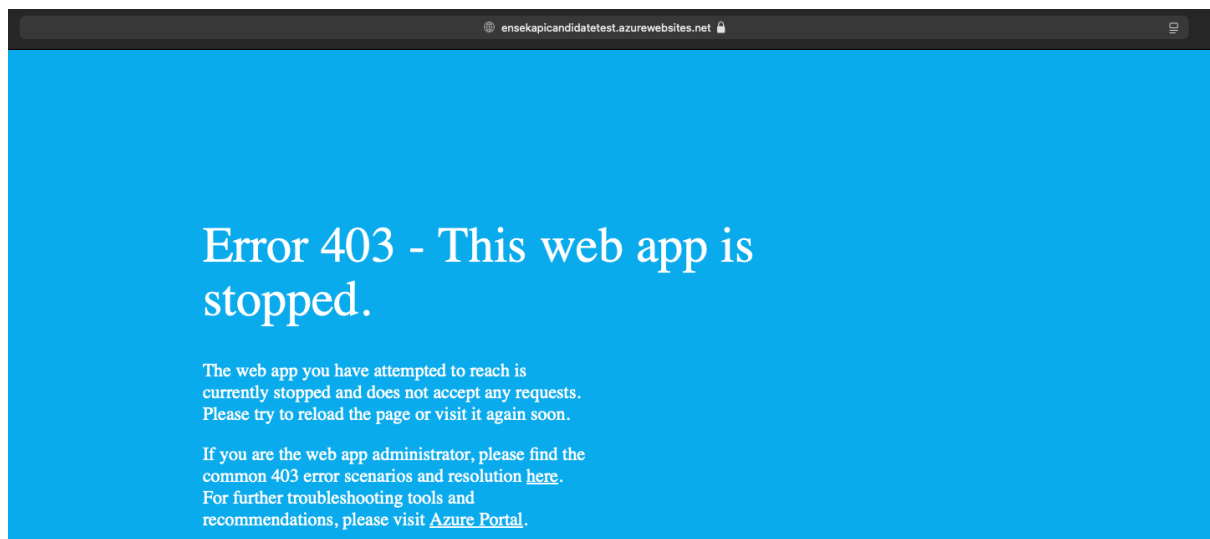


API Testing Summary: ENSEK Candidate Test API

Base URL: <https://ensekapicandidatetest.azurewebsites.net/>

API Access Status: “403 - Web App Stopped”

Despite multiple attempts to access this URL to view the Swagger UI and associated documentation, the application returned the following error:



This indicates that the hosted API service is currently unavailable, possibly due to the application being paused or the underlying App Service being deallocated in Azure.

Swagger Access Attempts

To rule out alternate Swagger routes, I attempted the following known documentation patterns based on the candidate API URL:

Attempts:

1. <https://ensekapicandidatetest.azurewebsites.net/swagger>
2. <https://ensekapicandidatetest.azurewebsites.net/swagger/index.html>
3. <https://ensekapicandidatetest.azurewebsites.net/swagger.json>
4. <https://ensekapicandidatetest.azurewebsites.net/api-docs>
5. <https://ensekapicandidatetest.azurewebsites.net/api/swagger>
6. <https://ensekapicandidatetest.azurewebsites.net/docs>
7. <https://ensekapicandidatetest.azurewebsites.net/help>
8. <https://ensekapicandidatetest.azurewebsites.net/openapi>
9. <https://ensekapicandidatetest.azurewebsites.net/openapi.json>

All returned either a 404 or 403 response, indicating that Swagger UI or OpenAPI documentation was not available at the time of testing.

Proposed Automation Testing Approach

The following approach outlines assumed endpoints and their intended behaviour. Test scenarios are written in Gherkin-style pseudocode, making them compatible with most Behaviour-Driven Development (BDD) frameworks such as Cucumber, Specflow, or Reqrroll. In a more production-oriented context, I would implement these tests by building a dedicated API test suite using tools like Postman or Hoppscotch, organizing requests into collections, and integrating the suite into a CI/CD pipeline to support continuous validation.

Scenario 1: Reset the test data

```
1 Feature: Reset application test data
2
3   Scenario: Successfully reset the state of the API
4     Given the API is available at https://ensekapicandidatetest.azurewebsites.net/reset
5     When I send a POST request to "/reset"
6     Then the response code should be 200
7     And the application state should be cleared to its initial condition
```

Expected Result:

HTTP Status Code: 200 OK

```
1 {
2   "status": "Reset successful"
3 }
```

Scenario 2: Buy a quantity of each fuel

```
1 Feature: Purchase available fuel types
2
3   Scenario Outline: Successfully purchase units of fuel
4     Given the API is available at https://ensekapicandidatetest.azurewebsites.net/buy
5     When I send a POST request to "/buy" with body:
6       | fuelType | quantity |
7       | <fuelType> | <qty> |
8     Then the response code should be 200
9     And the response should contain a valid order ID
10    And the response should confirm the correct fuelType and quantity
11
12    Examples:
13      | fuelType | qty |
14      | Gas      | 5   |
15      | Electric | 10  |
16      | Oil      | 3   |
```

Sample Request Payload:

```
1 {
2   "fuelType": "Gas",
3   "quantity": 5
4 }
```

Expected Result:

HTTP Status Code: 200 OK

Sample Response Body:

```
1 {
2   "orderId": 123,
3   "fuelType": "Gas",
4   "quantity": 5,
5   "timestamp": "2025-05-09T19:45:00Z"
6 }
```

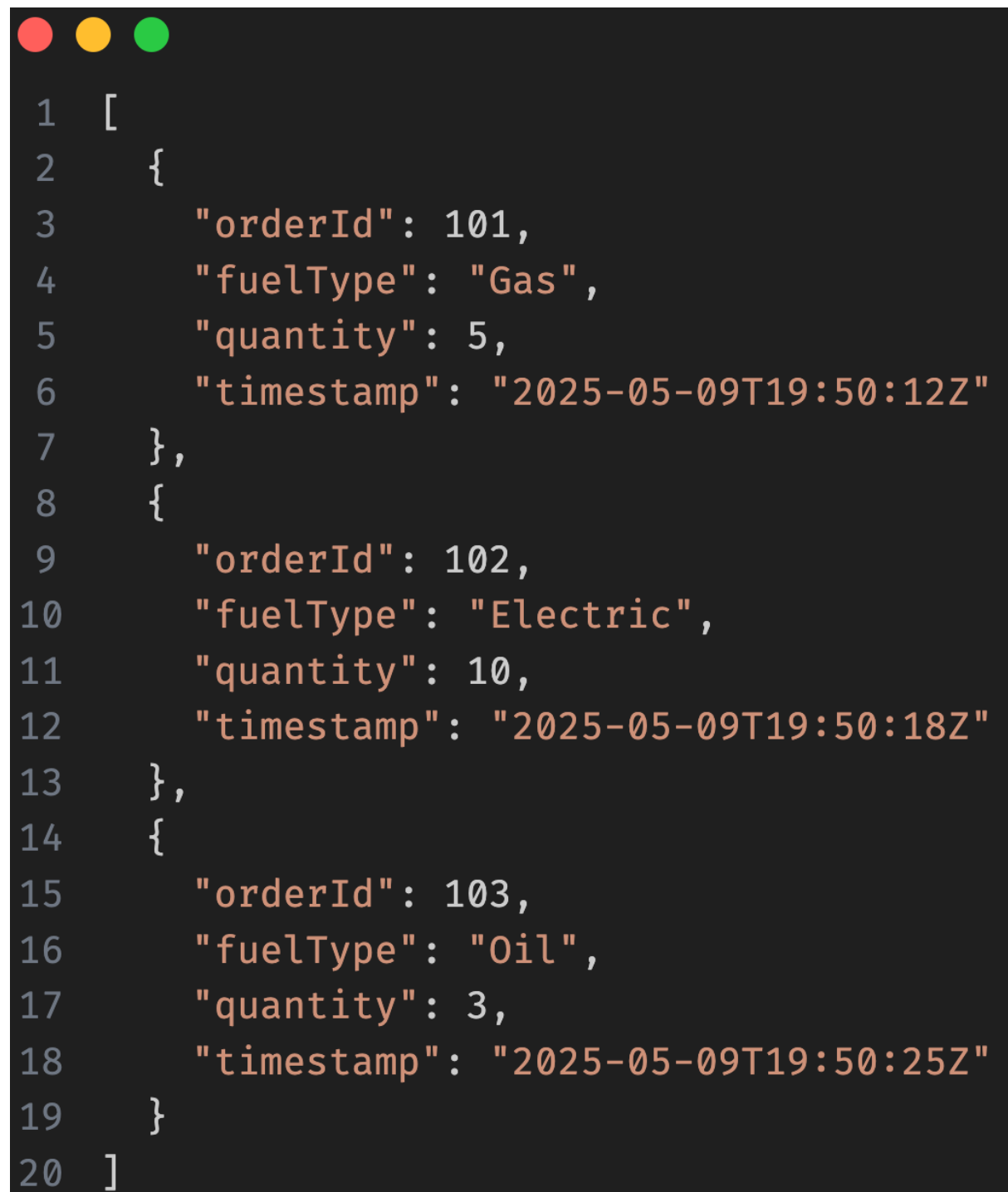
Scenario 3: Verify that each order from the previous step is returned in the /orders list with the expected details

```
1 Feature: Validate order persistence and correctness
2
3 Scenario: Confirm that all placed orders are listed correctly
4   Given I have successfully placed orders for Gas, Electric, and Oil
5   When I send a GET request to "/orders"
6   Then the response code should be 200
7   And the response should include entries for:
8     | fuelType | quantity |
9     | Gas      | 5        |
10    | Electric | 10       |
11    | Oil      | 3        |
12   And each order should contain:
13     - a non-null orderId
14     - a valid timestamp
15     - correct fuelType and quantity
```

Expected Result:

HTTP Status Code: 200 OK

Sample Response Body:



```
1  [  
2    {  
3      "orderId": 101,  
4      "fuelType": "Gas",  
5      "quantity": 5,  
6      "timestamp": "2025-05-09T19:50:12Z"  
7    },  
8    {  
9      "orderId": 102,  
10     "fuelType": "Electric",  
11     "quantity": 10,  
12     "timestamp": "2025-05-09T19:50:18Z"  
13   },  
14   {  
15     "orderId": 103,  
16     "fuelType": "Oil",  
17     "quantity": 3,  
18     "timestamp": "2025-05-09T19:50:25Z"  
19   }  
20 ]
```

Scenario 4: Confirm how many orders were created before the current date

NB* Assumption has been made that the response would be a plain object and not a count value but will support logical filtering by timestamp.

```
1 Feature: Count historical orders based on timestamp
2
3   Scenario: Determine number of orders placed before today
4     Given today's date is 2024-05-09
5     When I send a GET request to "/orders"
6     Then the response code should be 200
7     And I should receive a list of order objects
8     And I should be able to count how many have a timestamp before "2025-05-09T00:00:00Z"
```

Expected Result:

HTTP Status Code: 200 OK

Sample Response Body:

```
1  [
2    {
3      "orderId": 201,
4      "fuelType": "Gas",
5      "quantity": 10,
6      "timestamp": "2024-05-08T14:32:10Z"
7    },
8    {
9      "orderId": 202,
10     "fuelType": "Electric",
11     "quantity": 8,
12     "timestamp": "2024-05-08T08:45:00Z"
13   },
14   {
15     "orderId": 203,
16     "fuelType": "Oil",
17     "quantity": 4,
18     "timestamp": "2024-05-08T16:27:50Z"
19   }
20 ]
```

Additional Note – As no direct query or count endpoint appears to be exposed, it is assumed that the full list of orders would be returned via GET /orders. Any filtering or aggregation (e.g., total orders before a specific date) would therefore be handled on the client side or by a downstream processing service. This aligns with a common API design pattern where business-specific metrics are derived using filter and reduce operations over the full dataset.

Scenario 5: Additional Validation Tests for API Quality Assurance

```
1 Feature: Handle edge cases and robustness in order submission
2
3 Scenario Outline: Submit invalid or boundary fuel order requests
4   Given the API is available at https://ensekapicandidatetest.azurewebsites.net/buy
5   When I send a POST request to "/buy" with invalid or edge input:
6     | fuelType      | quantity  |
7     | <fuelType>     | <quantity> |
8   Then the response code should be appropriate:
9     - 400 for bad input
10    - 404 for unknown fuelType
11    - 422 for logical violations
12   And the response should contain a meaningful error message
13
14   Examples:
15     | fuelType      | quantity  |
16     | Gas           | -1        |
17     | Gas           | 0         |
18     | Gas           | "five"    |
19     | BananaFuel    | 3         |
20     | null          | 10        |
21     | Gas           | 99999999  |
```

Expected Sample Responses:

Test Case	Expected HTTP Status	Expected Error Message
fuelType: "Gas", quantity: -1	400 Bad Request	"Quantity must be positive"
fuelType: "Gas", quantity: 0	400 Bad Request	"Quantity must be greater than zero"
fuelType: "Gas", quantity: "five"	400 Bad Request	"Invalid quantity format"
fuelType: "BananaFuel", quantity: 3	404 Not Found	"Fuel type not recognized"
fuelType: null, quantity: 10	400 Bad Request	"Fuel type is required"
fuelType: "Gas", quantity: 99999999	422 Unprocessable Entity	"Quantity exceeds available stock"