# Automated API tests

This document gives a high-level explanation / overview of the automated API tests and my thinking behind them.

## GET Energy

* Just a simple check for 200 response.

## PUT – buy energy

These tests are the same for all energy types that have non-zero quantities available for purchase.

* Use of regular expressions to validate parts of the response message that are consistent throughout all requests.
* Use of regular expressions to pull out the unique order id and store it in an environment variable to be used in the GET orders endpoint tests.
* Also included a simple 200 response test.
* NOTE: I have not used regular expressions. I used generative AI to help me with the creating regex tests.

## GET Orders

* Tests to check if each order id for each energy type are present in the list.
* Test to check the number of orders placed before the current date.
* Simple 200 response test.

## POST – reset data

* Check that the response does not return unauthorised as I wasn’t sure if the correct response code for a successful request would return 200 or 201.
  + I probably could have included 200 or 201 check.
  + However, I noticed that I always get 401 despite being authorised. So this is a bug I have raised.

## Running the tests

* I have exported my Postman collection.
  + Re-import it into Postman if you want to look through the tests.
  + Or view the tests on the next page.
* I used Newman to run the tests automatically in the command line.
  + <https://learning.postman.com/docs/collections/using-newman-cli/installing-running-newman/#installing-newman>
* Install Newman globally: npm install -g newman
* In a command console, navigate to the folder containing the exported Postman collection
* Run the following command: newman run ENSEK-TEST.postman\_collection

GET energy tests:  
pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

### PUT – buy oil energy tests:

pm.**test**("Validate purchase response message", **function** () {

    var responseJson **=** pm.response.json();

    var message **=** responseJson.message;

    // Regular expression to validate the message format

    var regex **=** /You have purchased \d**+** Litres at a cost of [\d.]**+** there are \d**+** units remaining\. Your order\s**+**id is [\w-]**+**./;

    // note: this test will fail due to a grammatical erorr in the repsone "Your orderid" should be "Your order id"

    pm.expect(regex.**test**(message)).to.be.true, "Response message does not match expected format";

    // Extract and store the order ID for further tests

    var oilOrderIdRegex **=** /Your order\s**+**id is ([\w-]**+**)/;

    var oilOrderIdMatch **=** oilOrderIdRegex.**exec**(message);

    pm.expect(gasOrderIdMatch).to.not.be.null, "Order ID not found in message";

**if** (oilgasOrderIdMatch **&&** oilgasOrderIdMatch.length **>** 1) {

        var oilOrderId **=** oilOrderIdMatch[1];

        pm.environment.**set**("oilRecentOrderId", oilOrderId);

    }

});

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

### PUT – buy electricity energy tests:

pm.**test**("Validate purchase response message", **function** () {

    var responseJson **=** pm.response.json();

    var message **=** responseJson.message;

    // Log the response message for debugging

    // console.log("Response message: " + message);

    // Regular expression to validate the message format

    var regex **=** /You have purchased \d**+** kWh at a cost of [\d.]**+** there are \d**+** units remaining\. Your order\s**+**id is [\w-]**+**./;

    pm.expect(regex.**test**(message)).to.be.true, "Response message does not match expected format";

    // Extract and store the order ID for further tests

    var electricityOrderIdRegex **=** /Your order\s**+**id is ([\w-]**+**)/;

    var electricityOrderIdMatch **=** electricityOrderIdRegex.**exec**(message);

    pm.expect(electricityOrderIdMatch).to.not.be.null, "Order ID not found in message";

**if** (electricityOrderIdMatch **&&** electricityOrderIdMatch.length **>** 1) {

        var electricityOrderId **=** electricityOrderIdMatch[1];

        pm.environment.**set**("recentOrderId", electricityOrderId);

        console.**log**(electricityOrderId)

    }

});

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

### PUT – buy gas energy tests:

pm.**test**("Validate purchase response message", **function** () {

    var responseJson **=** pm.response.json();

    var message **=** responseJson.message;

    // Log the response message for debugging

    // console.log("Response message: " + message);

    // Regular expression to validate the message format

    var regex **=** /You have purchased \d**+** m³ at a cost of [\d.]**+** there are \d**+** units remaining\. Your order\s**+**id is [\w-]**+**./;

    pm.expect(regex.**test**(message)).to.be.true, "Response message does not match expected format";

    // Extract and store the order ID for further tests

    var gasOrderIdRegex **=** /Your order\s**+**id is ([\w-]**+**)/;

    var gasOrderIdMatch **=** gasOrderIdRegex.**exec**(message);

    pm.expect(gasOrderIdMatch).to.not.be.null, "Order ID not found in message";

**if** (gasOrderIdMatch **&&** gasOrderIdMatch.length **>** 1) {

        var gasOrderId **=** gasOrderIdMatch[1];

        pm.environment.**set**("gasRecentOrderId", gasOrderId);

        console.**log**(gasOrderId)

    }

});

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});

### PUT – buy nuclear energy tests:

pm.**test**("Validate purchase response message", **function** () {

    var responseJson **=** pm.response.json();

    var message **=** responseJson.message;

    pm.expect(message).to.eql("There is no nuclear fuel to purchase!");

  // If I had more time, I'd update with test with if/else statements to cover the scenarios where there is and isn't fuel available.

});

pm.**test**("Status code is 200", **function** () {

    pm.response.to.have.status(200);

});