Domain:

```
domain = 'https://www.*******.uk'
```

Questions

- Why in Calculation results "Volume" and "1 mln USD" always in USD?
- What logic do complex conversion use?
- Do "EURAUD" and "AUDEUR" have the same conversion price? Where it goes from?
- Entry point got DDoS protection? 20 requests in 20 sec detects as flood attack?
- When I try to get price at, for example 23:28, I got no data. The reason is the market is closed or something
  doesn't work properly? Don't we need some additional info on frontend page about it? (this could be connected with
  security lock, see bugs block)
- What role gets "leverage" in this calculations? I couldn't find connection with result.
- Why data validation fail methods return response code 200?
- How does frontend client language selector works? I have russian warning on desktop, but no warning on Android phone.

Frontend test cases:

link
Do not forget to open console and check js errors!

Environment:

- Desktop
- Google chrome (up to date)

Positive cases

Region test:

- Change IP location to European country

- Open https://www.*******.com/tools/calculator/

Expected result:

- redirect to .uk zone

Adaptive design:

- Open link
- Change browser window to small and middle sizes

Expected result:

- page design is adaptive and look nice

Compatibility:

- Open link in mobile browser

Expected result:

- page design look readable and nice
- Clock "Calculate"

Expected result:

- Two blocks: "Calculation results" an "Calculation formulas" filled by some data
- Repeat for up-to-date Firefox and Safari

Smoke positive:

- Open link
- Click "Calculate"

Expected result:

- "Calculation results" block:
  - all values are filled
  - all currency is equal to "Account currency", except "Volume" and "1 mln USD"
- "Calculation formulas" block:

    - "Margin", "Point profit", "Volume" have one "formula" line and one calculation line
    - "Swap" has one "formula" line and two calculation line
    - Formulas calculation are correct and not rounded
    - "Lots" value is same, as in input form
    - "Margin", "Volume" currency is first one from "Forex" input value
    - "Point profit", "Swap" currency is second one from "Forex" input value
    - "Currency pairs used during conversion" filled with only currencies, used in "Forex" and "Account currency" input values

- Remember current values somehow
- Random change all input values
- Values on input form match changed values
- Click "Calculate"
- Expected result differ, but all previous validations pass

Input filters

- Open link
- dropdown lists, current currency is typed gray and with "check" mark, "hover" line is colored red:
    - "Account type" field with one value: "Pro"
    - "Exchange" field with one value: "Forex - Foreign exchange"
    - "Forex" field with range of 6 char currencies list, scroll works
    - "Leverage" field has values from list {1:200, 1:100, 1:88, 1:50, 1:20, 1:10, 1:2}
    - "Account currency" has list of currencies with text descriptions
- "Lot" value could be only in range [0.01, 100000], small arrows in field inc/dec value by step 0.01

Currency:

common_currencies = {'CHF', 'EUR', 'AUD', 'CAD', 'DKK', 'GBP', 'HUF', 'JPY', 'MXN', 'NOK', 'NZD', 'PLN', 'SEK', 'SGD', 'TRY', 'USD', 'ZAR'}

- Open link
- Select "Account currency" and "Forex" values to match in one value.
- Click "Calculate"

Expected result:

- "Currency pairs" block has one currency and it's same as "Forex" value
- Select diverse "Account currency" and "Forex" values, but "Account currency" must be in common_currencies set and "Forex" without "USD".
- Click "Calculate"

Expected result:

- "Currency pairs" block has two values:
  - "Account currency" + "Forex" value first 3 chars (parts order could differ)
  - "Account currency" + "Forex" value last 3 chars (parts order could differ)
- Select diverse "Account currency" and "Forex" values, but "Account currency" must be in common_currencies set and
  "Forex" value have "USD" in it.
- Click "Calculate"

Expected result:

- "Currency pairs" block has two values:
  - "Account currency" + "Forex" value first 3 chars (parts order could differ)
  - "Account currency" + "Forex" value last 3 chars (parts order could differ)
- Select diverse "Account currency" and "Forex" values, but "Account currency" must be NOT in common_currencies set
  and "Forex" without "USD".
- Click "Calculate"

Expected result:

- "Currency pairs" block has three values:
  - "Account currency" + "USD" (parts order could differ) # if "Account currency" is 6-digit value, then this line'll
    be second, not first
  - "USD" + "Forex" value first 3 chars (parts order could differ)
  - "USD" + "Forex" value last 3 chars (parts order could differ)

Negative cases:

Leverage

- Open link
- Set "Leverage" to any text
- Click "Calculate"

Expected result:

- "Leverage" switched to default "0.01" value

Backend API method description:

HTTP Method: GET
Suburl: api/calculator/calculate/
Request headers: Accept: application/json, text/javascript, /; q=0.01
Success response code: 200
Values [frontend default params]:

- form_type [classic]
- instrument [Forex]
- symbol [AUDCAD]
- lot [0.1]
- leverage [200]
- user_currency [USD]

Additional info:

This method is blackbox for us, so without communication with business or tech docs we can only validate result format
(JSON schema), mistypes in formulas and partly input data processing logic.

JSON schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
```

```
"commission": {
  "type": "null"
},
"long": {
  "type": "string"
},
"lots_mln_usd": {
  "type": "string"
},
"margin": {
  "type": "string"
},
"margin_formula1": {
  "type": "string",
  "enum": ["Lots x Contract size x Required margin"]
},
"margin_formula2": {
  "type": "string"
},
"no_quotes": {
  "type": "boolean"
},
"profit": {
  "type": "string"
},
"profit_formula1": {
  "type": "string",
  "enum": ["Point size x Contract size x Lots"]
},
"profit_formula2": {
  "type": "string"
},
"short": {
  "type": "string"
},
"swap_char": {
  "type": "string"
},
"swap_enable": {
  "type": "boolean"
},
"swap_formula1": {
  "type": "string",
  "enum": ["Lots x Contract size x Short_or_Long x Point size"]
},
"swap_formula2": {
  "type": "string"
```

```
},
"swap_formula3": {
  "type": "string"
},
"swap_long": {
  "type": "string"
},
"swap_short": {
  "type": "string"
},
"tick_size": {
  "type": "integer"
},
"user_currency": {
  "type": "string"
},
"volume_formula1": {
  "type": "string",
  "enum": ["Lots x Contract size"]
},
"volume_formula2": {
  "type": "string"
},
"volume_mln_usd": {
  "type": "string"
},
"form_type": {
  "type": "string",
  "enum": ["classic"]
}
},
"required": [
"commission",
"conversion_pairs",
"long",
"lots_mln_usd",
"margin",
"margin_formula1",
"margin_formula2",
"no_quotes",
"profit",
"profit_formula1",
"profit_formula2",
"short",
"swap_char",
"swap_enable",
"swap_formula1",
```

```
  "swap_formula2",
  "swap_formula3",
  "swap_long",
  "swap_short",
  "tick_size",
  "user_currency",
  "volume_formula1",
  "volume_formula2",
  "volume_mln_usd",
  "form_type"
  ]
}
```

Response example

response code = 200

```
{
  "commission": null,
  "conversion_pairs": {
  "AUDUSD": 0.71853,
  "USDAED": 3.67296,
  "USDCAD": 1.3279
  },
  "long": "-0.33",
  "lots_mln_usd": "14.59",
  "margin": "1319.57",
  "margin_formula1": "Lots x Contract size x Required margin",
  "margin_formula2": "0.10 x 100000.0 x 5.0% = 500.000 AUD",
  "no_quotes": false,
  "profit": "2.77",
  "profit_formula1": "Point size x Contract size x Lots",
  "profit_formula2": "0.00010 x 100000.0 x 0.10 = 1.00 CAD",
  "short": "-0.43",
  "swap_char": "pt.",
  "swap_enable": true,
  "swap_formula1": "Lots x Contract size x Short_or_Long x Point size",
  "swap_formula2": "0.10 x 100000.0 x -0.11840 x 0.00010 = -0.12 CAD",
  "swap_formula3": "0.10 x 100000.0 x -0.15490 x 0.00010 = -0.15 CAD",
  "swap_long": "-0.12",
  "swap_short": "-0.15",
  "tick_size": 0,
  "user_currency": "AED",
  "volume_formula1": "Lots x Contract size",
  "volume_formula2": "0.10 x 100000 = 10000.00 AUD",
```

```
  "volume_mln_usd": "0.0072",
  "form_type": "classic"
}
```

Common response validation regExes and formulas:

req - request data array
resp - response data array

```
def str_to_float(s: str, no_exception: bool = True):
  try:
    float(s)
  except ValueError as e:
    if no_exception:
      return 0.0
    raise ValueError(e)


contract_size=100000.0
required_margin=0.05
point_size=0.00010
req_lot = str_to_float(req['lot'])

margin_formula2 = f"{req_lot} x {contract_size}  x {required_margin*100}% = {req
profit_formula2 = f"{point_size} x {contract_size} x {req_lot} = {point_size*con

pattern = f"{req_lot} x {contract_size} x -(\d{0,10}\.{0,1}\d{0,10}) x {point_si
swap_long_detailed = re.findall(pattern, resp['swap_formula2'])[0]  # must fail,
swap_formula2 = f"{req_lot} x {contract_size} x -{round(float(swap_long_detailed

pattern = f"{req_lot} x {contract_size} x -(\d{0,10}\.{0,1}\d{0,10}) x {point_si
swap_long_detailed = re.findall(pattern, resp['swap_formula3'])[0]  # must fail,
swap_formula3 = f"{req_lot} x {contract_size} x -{round(float(swap_long_detailed

volume_formula2 = f"{req_lot} x {contract_size} = {req_lot*contract_size} req['s
```

Available values:

Request for actual dictionaries data

```
currencies = {"AED", "AUD", "ARS", "AZN", "BDT", "BHD", "BND", "BRL", "CAD", "CH

symbols = {"AUDCAD", "AUDCHF", "AUDGBP", "AUDJPY", "AUDNZD", "AUDUSD", "CADCHF",
```

```
leverages = {200, 100, 88, 50, 20, 10, 2}

# got this limits from web filters, must be reviewed with business
lot_val_max = 100000
lot_val_min = 0.01
```

Test cases:

Positive cases:

Default web values

request: api/calculator/calculate/
data:

- form_type = classic
- instrument = Forex
- symbol = AUDCAD
- lot = 0.1
- leverage = 200
- user_currency = USD

response:

- scheme validation
- common regexes and formulas validation

Default web values (periodicity)

Save previous case result and execute same request again.
Results must be equal, char to char.

```
json.dumps(req1, sort_keys=True) == json.dumps(req, sort_keys=True)
```

Boundary Values

Cases for max and min values of lot and leverage

request: api/calculator/calculate/

data:

- form_type = classic
- instrument = Forex
- symbol = random.choice(symbols)
- lot = lot_val_max
- leverage = min(leverages)
- user_currency = random.choice(currencies)

data:

- form_type = classic
- instrument = Forex
- symbol = random.choice(symbols)
- lot = lot_val_min
- leverage = min(leverages)
- user_currency = random.choice(currencies)

data:

- form_type = classic
- instrument = Forex
- symbol = random.choice(symbols)
- lot = lot_val_max
- leverage = min(leverages)
- user_currency = random.choice(currencies)

data:

- form_type = classic
- instrument = Forex
- symbol = random.choice(symbols)
- lot = lot_val_max
- leverage = max(leverages)
- user_currency = random.choice(currencies)

response:

- scheme validation

- common regexes and formulas validation

Combinations

**We could get three combinations of response:

- user_currency in symbol - we got **one** item in conversion_pairs
- user_currency not in symbol, but it's common currency and could be not converted twice - we got **two** items in conversion_pairs
- user_currency not in symbol and not common - we got **three** items in conversion_pairs
  (if symbol got "USD" in it,
  we got **two** items instead)

run test for each conversation_pairs_len in [1, 3]

```
import random


common_currencies = {'CHF', 'EUR', 'AUD', 'CAD', 'DKK', 'GBP', 'HUF', 'JPY', 'MXI
uncommon_currencies = currencies - common_currencies


def random_currency(symbol: str, conversation_pairs_len: int): str
  """
  This algorithm was based on current method realisation, it could be wrong

  symbol - two currencies conversion string
  conversation_pairs_len - how many conversions we'd like to make [1,3]
  return random currency with selected conversation pair
  """

  selected_currencies = {symbol[0:3], symbol[3:6]}

  if 1 == conversation_pairs_len:
    return random.choice(selected_currencies)
  else:
    uncrossed_currencies = common_currencies - selected_currencies
    if 2 == conversation_pairs_len:
      return random.choice(uncrossed_currencies)
    else:
      return random.choice(uncommon_currencies - selected_currencies)


def random_lot(): float
```

```
    return random.randint(lot_val_min*100, lot_val_max*100) / 100  # could be bad
```

request: api/calculator/calculate/
data:

- form_type = classic
- instrument = Forex
- symbol = random.choice(symbols)
- lot = random_lot()
- leverage = random.choice(leverages)
- user_currency = random_currency(symbol, conversation_pairs_len)

response:

- scheme validation
- common regexes and formulas validation
- len(conversion_pairs) validation

Negative cases:

There is no documentation for this API, so we try to guess response result and made to improvise.
Logic could differ from original logic, all expected result must be discussed with business.

Bad header Accept

request: http://{domain}/api/calculator/calculate/
header:
Accept: text/html
data:

- form_type = classic
- instrument = Forex
- symbol = AUDCAD
- lot = 0.1
- leverage = 200
- user_currency = USD

response:

status_code != 200

Empty request

request: api/calculator/calculate/
data:

response:
status_code != 200


```
{
  "form_type": [
  "This field is required."
  ],
  "lot": [
  "This field is required."
  ],
  "leverage": [
  "This field is required."
  ],
  "instrument": [
  "This field is required."
  ],
  "symbol": [
  "This field is required."
  ],
  "user_currency": [
  "This field is required."
  ]
}
```


Uncompleted data

Do not send one of data values from "data" section

request: api/calculator/calculate/
data:

- form_type = classic
- instrument = Forex
- symbol = AUDCAD
- lot = 0.1
- leverage = 200

- user_currency = USD

response:
status_code != 200
example:

```
{
    'user_currency': ['This field is required.']
}
```

Wrong valid type data

Use wrong but valid type data
Run request for every value in list, while other values stay valid.

Run request for each "form_type" in ["ClassiC", "A", ""]
Run request for each "instrument" in ["foREX", "A", ""]
Run request for each "symbol" in ["AudcaD", "RURRUR", ""]
Run request for each "lot" in [0, -1, 100001, 99999.9999999]
Run request for each "leverage" in [0, -1, 300, 200.0]
Run request for each "user_currency" in ["UsD", "ZZZ", "A", ""]

request: api/calculator/calculate/
data:

- form_type = classic
- instrument = Forex
- symbol = AUDCAD
- lot = 0.1
- leverage = 200
- user_currency = USD

response:
status_code != 200 # if status_code == 200, validate schema and formulas and review test case
example:

```
{
    'user_currency': ['"RURRUR" is not a valid choice.']
}
```

Unsupported data

Use boundary values, but out of order, unsupported values.
Replace Null with None, null, etc, if you got something more specific about backend.

Run request for each value in [-1, 0.0, Null, , "Я", ".", "#", "' OR 1=1 -- ", "ABCDEFGHIJKLMNOPQSTUVWXYZABCDEFGHIJKLMNOPQSTUVWXYZABCDEFGHIJKLMNOPQSTUVWXYZABCDEFGHIJKLMNOPQSTUVWXYZABCDEFGHIJKLMNOPQSTUVWXYZABCD"], while other values stay valid.

request: api/calculator/calculate/
data:.

- form_type = classic
- instrument = Forex
- symbol = AUDCAD
- lot = 0.1
- leverage = 200
- user_currency = USD

response:
status_code != 200 # if status_code == 200, validate schema and formulas and review test case