Consideraciones

Diagrama de base de datos:

https://drive.google.com/file/d/1IxUE9MCKJeSbKLi_jSGbS-X1ZxZ_oZ5e/view

Yo utilize mongoDB por facilidad en las consultas, pero este esquema se puede implementar en DynamoDB con un buen diseño de partitionKey y sortKey

Nomenclatura base de datos:

Para la nomenclatura de la base de datos se siguió el siguiente estándar:

https://tennant.io/quick-mongodb-naming-conventions/

Selección de Targets:

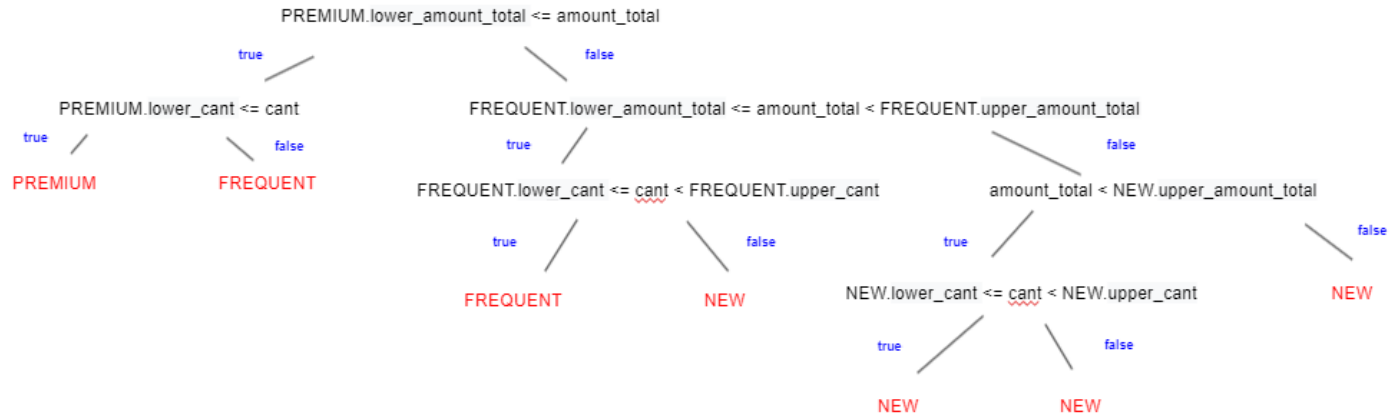
Yo identifique que por como se definieron los targets pueden haber ciertas inconsistencias, ya que suponiendo que en el 2021 se una persona fue target NEW y se sacó solamente un crédito de 300000, si en el siguiente año quisiera sacar otro crédito, por el número de créditos tomados en el 2021 y en el amount, si se siguieran las condiciones del documento para seleccionar el target de la persona al pie de la letra esta persona no entraría en ningún target, ya que para NEW cumple la de cant (1) pero no cumple la del amount_total (300000), y para FREQUENT no cumple con la de cant (1) pero cumple con la del del amount_total (300000).

Por lo que para poder solucionar este problema de que se cumple una condición per no se cumple otra y que el usuario no pueda encajar en ningún target yo decidí implementar un árbol de decisión, y en el caso de que se cumpla una y no la otra se le asignará un target de acuerdo a la estructura de mi arbol de decision (una suposicion mia).

En base de datos se va a guardar la tasa y el máximo a retirar por préstamo por cada target. Y se va a guardar un array de json que contiene el esquema del árbol de decisión con el cual se va a poder determinar por medio del amount_total y cant el target de la persona.

Por como está definido el arbol los casos que no entran en ninguna de las condiciones del documento como por ejemplo cant 1 y amount 500000 entrarian a un target definido a mi criterio

Imagen de arbol de decision implementado:



Por lo anterior el usuario como tal no está relacionado a un target sino que el target se calcula en el momento que el usuario va a sacar un crédito

Forma en la que se trabajó la deuda hasta la fecha:

Deuda hipotetica:

Si una persona saca un credito con un amount de 100000 a 3 meses el 2022-01-01 para este ejercicio la deuda total será de 102510.36 añadiendo los intereses y estaría repartida en 3 intervalos de tiempo:

1. Mes 1:
   2022-01-01 hasta 2022-01-31
   Cuota: 34170.12

2. Mes 2:
   2022-02-01 hasta 2022-02-28
   Cuota: 34170.12

3. Mes 3:
   2022-03-01 hasta 2022-03-31
   Cuota: 34170.12

Por lo que la deuda hipotética en cada mes va a ser el acumulado de la deuda hasta ese punto:

Mes 1: 34170.12

Mes 2: 68340.24

Mes 3: 102510.36

Deuda Real:

La deuda real es básicamente la deuda hipotética en un punto del tiempo restando lo que se ha pagado en su totalidad hasta esa fecha (Acumulado), si la resta es negativa la deuda será 0:

Para este escenario solo se hizo el pago del primer mes y se pagó menos que la cuota (En el documento se especificó que esto era posible)

1. Mes 1:
   2022-01-01 hasta 2022-01-31
   Deuda hipotetica: 34170.12
   Pago: 34000
   Acumulado: 34000

2. Mes 2:
   2022-02-01 hasta 2022-02-28
   Deuda hipotetica: 68340.24
   Pago: Pendiente
   Acumulado: 34000

3. Mes 3:
   2022-03-01 hasta 2022-03-31
   Deuda hipotetica: 102510.36
   Pago: Pendiente
   Acumulado: 34000

Por lo tanto la deuda real en cada mes va a ser la siguiente:

Mes 1: 170

Mes 2: 34,340.12

Mes 3: 68,510.24

Por lo que siguiendo con este ejemplo vamos a hacer que en el segundo mes se pague mucho más que la cuota de ese mes (En el documento se especificó que eso era posible)

1. Mes 1:

2022-01-01 hasta 2022-01-31
Deuda hipotetica: 34170.12
Pago: 34000
Acumulado: 34000

2. Mes 2:
   2022-02-01 hasta 2022-02-28
   Deuda hipotetica: 68340.24
   Pago: 60000
   Acumulado: 94000

3. Mes 3:
   2022-03-01 hasta 2022-03-31
   Deuda hipotetica: 102510.36
   Pago: Pendiente
   Acumulado: 94000

Solo se actualizan los acumolados desde el mes del pago en adelante, por lo que después de se pagó la deuda real en cada mes va a ser la siguiente:

Mes 1: 170

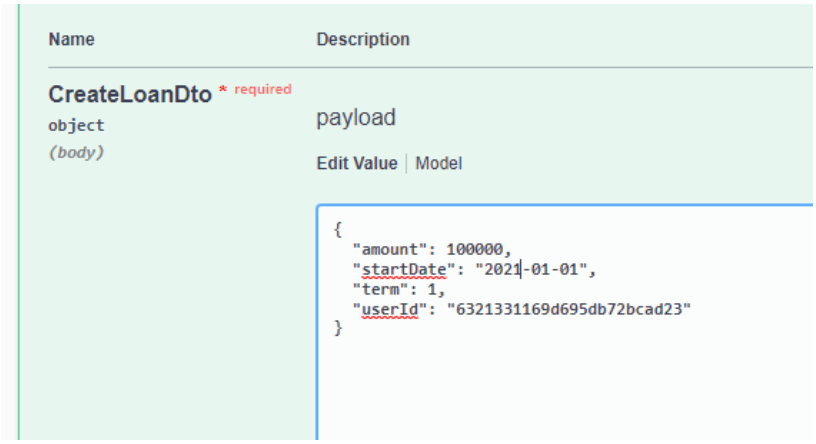
Mes 2: 0

Mes 3: 8510.36

Como no se actualiza el acumulado del mes anterior la deuda del mes anterior no va a cambiar. Y eso permite poder saber en el histórico de un crédito si la persona en algún punto estuvo en deuda.

Creación de un crédito:

Para este ejercicio y por definición del arbol de decision usuarios sin creditos ni amount van a poder sacar cretidos y por defecto su target sera NEW, Pero en si no hay ninguna limitación desde que el amount del credito a tomar sea acorde al del target, **por lo que un usuario va a poder sacar todos los creditos que quiera en esta aplicacion independiente** si esta endeudado o no.



Yo hice que el endpoint reciba la fecha de inicio del crédito para facilidad de pruebas por lo que para probar que lo de los targets funciona se puede crear 2 creditos a un usuario nuevo de term 1 y amount 100000 en el 2021 los cuales seran de target NEW ya que el usuario es nuevo, y después crear un credito el 2022, y este nuevo credito del 2022 ya sera FREQUET porque el usuario en el 2021 tuvo 2 creditos

El target se define por los créditos del año anterior al año del nuevo credito a sacar


```
{
  "id": "6325e1e600bedd601ce852d0",
  "amount": 100000,
  "term": 3,
  "rate": 0.15,
  "userId": "6321331169d695db72bcad23",
  "targetSchemaId": "6321162b88d6e09094f568ba",
  "targetName": "NEW",
  "startDate": "2022-01-01T00:00:00Z",
  "endDate": "2022-03-31T00:00:00Z",
  "quota": 34170.12,
  "debt": 102510.36,
  "loanHistory": [
    {
      "monthStart": "2022-01-01T00:00:00Z",
      "monthEnd": "2022-01-31T00:00:00Z",
      "accumulated": 0,
      "monthDebt": 34170.12,
      "paymentId": "000000000000000000000000"
    },
    {
      "monthStart": "2022-02-01T00:00:00Z",
      "monthEnd": "2022-02-28T00:00:00Z",
      "accumulated": 0,
      "monthDebt": 68340.24,
      "paymentId": "000000000000000000000000"
    },
    {
```


Al crear un credito yo automáticamente generó el histórico de cada mes desde la fecha de inicio, con la deuda hipotética y el acumulado hasta ese mes

Pago de una cuota de un crédito:

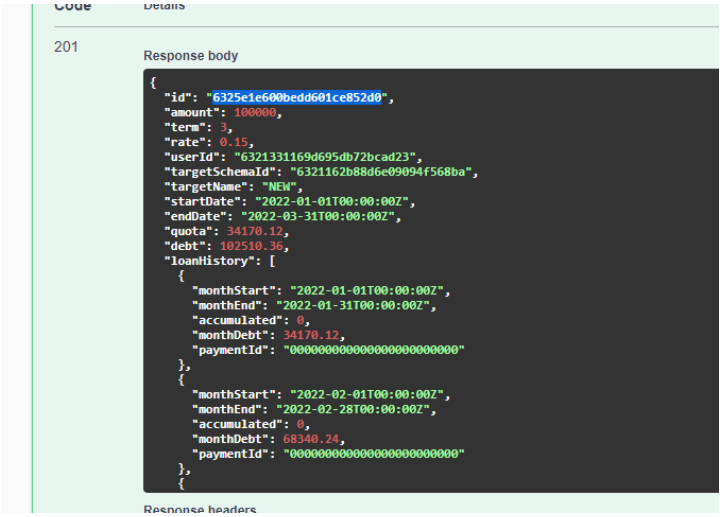
El endpoint para pagar en mi caso lo le permití que se recibiera la fecha del mes del que se desea hacer un pago, por lo que si se intenta hacer un pago en una fecha que está fuera del credito el endpoint va a tirar un error, ahora si es una fecha dentro del crédito, se va a buscar esa fecha a que mes del histórico corresponde, actualiza el acumulated del mes del pago, y de todos los meses siguientes,no los anteriores, crea el pago, y le resta lo pagado a la deuda total del crédito.

Si en el histórico un mes ya esta pagado (tiene id de pago) este ya no se va a poder volver a pagar por lo que si una persona en este ejercicio todos los meses paga menos va a quedar con deuda para siempre

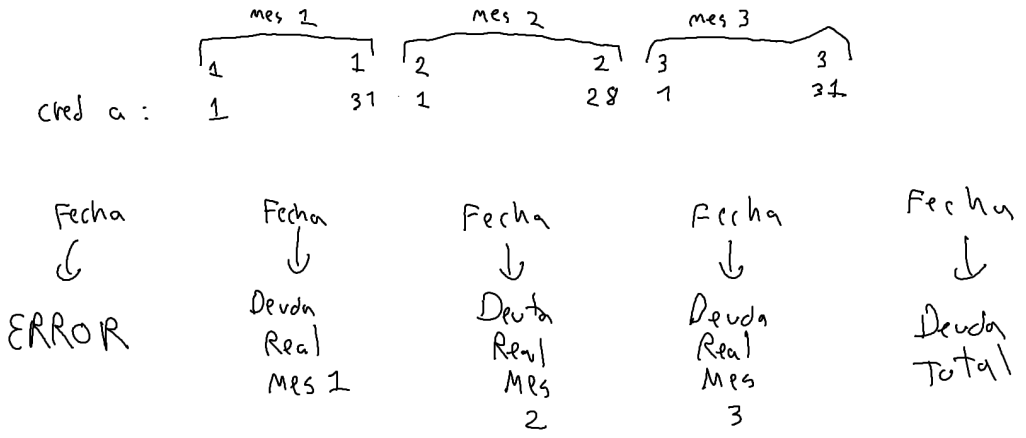
Por el diseño que yo hice, si bien es posible almacenar la información del pago en el histórico directamente, yo preferí tener una tabla separada para mayor flexibilidad de consultas a futuro.

Cálculo de la deuda de un crédito hasta una fecha:

Para mi caso yo al crear un crédito guardo siempre la fecha de inicio y de fin, todo el histórico con los acumulados en 0:

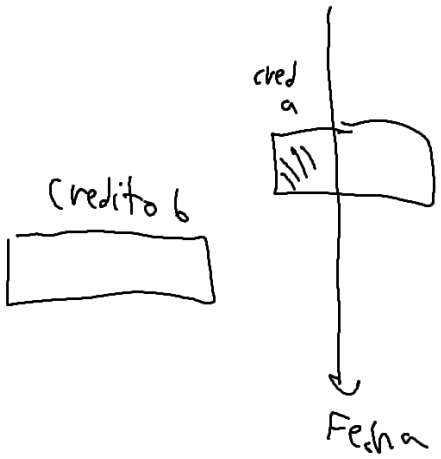


Por lo que si la fecha pasada es antes de que el crédito inicie el back arrojará un error, si la fecha está dentro del rango de alguno de los meses del histórico del crédito se retornara la deuda real de ese mes (Deuda hipotética - Acumulado de ese mes), y si la fecha está después de la fecha de fin del crédito se retornara la **deuda total**.



Cálculo de la deuda de todos los créditos:

Para este caso si no se envía una fecha se va a calcular la **deuda total** de todos los créditos donde debt != 0, ahora si se envía una fecha se va a calcular la deuda de todos los créditos iniciados **antes** de esa fecha o iniciados en esa fecha, lo que nos deja con 2 escenarios, si la fecha de fin del crédito está antes de la fecha de la consulta o después.
Si la fecha de fin del credito está después de la fecha pasada es porque la fecha cayó en la mitad de un crédito por lo que se calculará la deuda real del crédito en el mes donde cayó la fecha, si la fecha de fin del credito tambien esta antes de la fecha pasada, simplemente se retornara la deuda total del crédito.



Datos de prueba creados a montar el proyecto:

Al correr el docker compose up en la base de datos se va a crear lo siguiente:

Va a crear 2 usuarios:

```
[
  {
    "id": "63262ccaf944867773f46389",
    "name": "Juan Pablo"
  },
  {
    "id": "63262dd9f944867773f4638a",
    "name": "Carlos"
  }
]
```

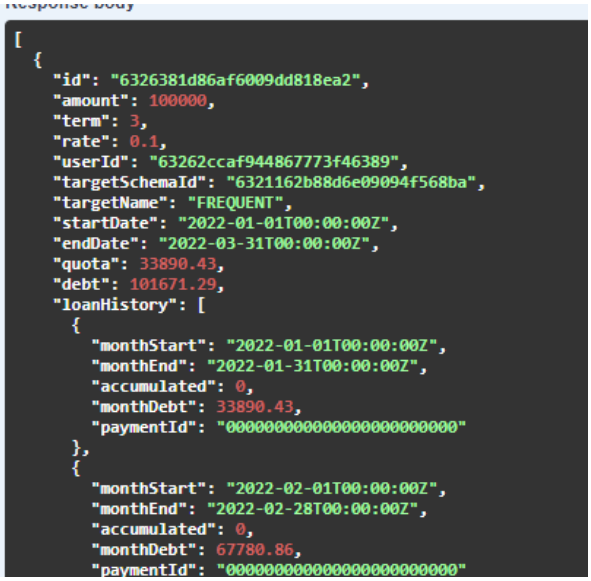
El usuario Juan Pablo va a tener 2 créditos de una cuota tomados el 2021 y pagados en su totalidad de target NEW, por lo que van a estar los pagos de esos créditos, Y por consiguiente si el usuario Juan Pablo saca créditos en el 2022 el target de estos será FREQUENT

```
[
  {
    "id": "6326343e715158faa6f71d4e",
    "amount": 100000,
    "term": 1,
    "rate": 0.15,
    "userId": "63262ccaf944867773f46389",
    "targetSchemaId": "6321162b88d6e09094f568ba",
    "targetName": "NEW",
    "startDate": "2021-01-01T00:00:00Z",
    "endDate": "2021-01-31T00:00:00Z",
    "quota": 101250,
    "debt": 0,
    "loanHistory": [
      {
        "monthStart": "2021-01-01T00:00:00Z",
        "monthEnd": "2021-01-31T00:00:00Z",
        "accumulated": 101250,
        "monthDebt": 101250,
        "paymentId": "6326344b715158faa6f71d4f"
      }
    ]
  },
  {
    "id": "6326345b715158faa6f71d50",
    "amount": 100000,
    "term": 1,
    "rate": 0.15
```

También se crearán 2 créditos nuevos de 3 cuotas en el 2022 sin ningun pago:

Credito Juan Pablo:

Rango: 2022-01-01 - 2022-03-31
Target: Frequent



```
[
  {
    "id": "6326381d86af6009dd818ea2",
    "amount": 100000,
    "term": 3,
    "rate": 0.1,
    "userId": "63262ccaf944867773f46389",
    "targetSchemaId": "6321162b88d6e09094f568ba",
    "targetName": "FREQUENT",
    "startDate": "2022-01-01T00:00:00Z",
    "endDate": "2022-03-31T00:00:00Z",
    "quota": 33890.43,
    "debt": 101671.29,
    "loanHistory": [
      {
        "monthStart": "2022-01-01T00:00:00Z",
        "monthEnd": "2022-01-31T00:00:00Z",
        "accumulated": 0,
        "monthDebt": 33890.43,
        "paymentId": "000000000000000000000000"
      },
      {
        "monthStart": "2022-02-01T00:00:00Z",
        "monthEnd": "2022-02-28T00:00:00Z",
        "accumulated": 0,
        "monthDebt": 67780.86,
        "paymentId": "000000000000000000000000"
```


Credito Carlos:

Rango: 2022-02-01 - 2022-04-30
Target: New

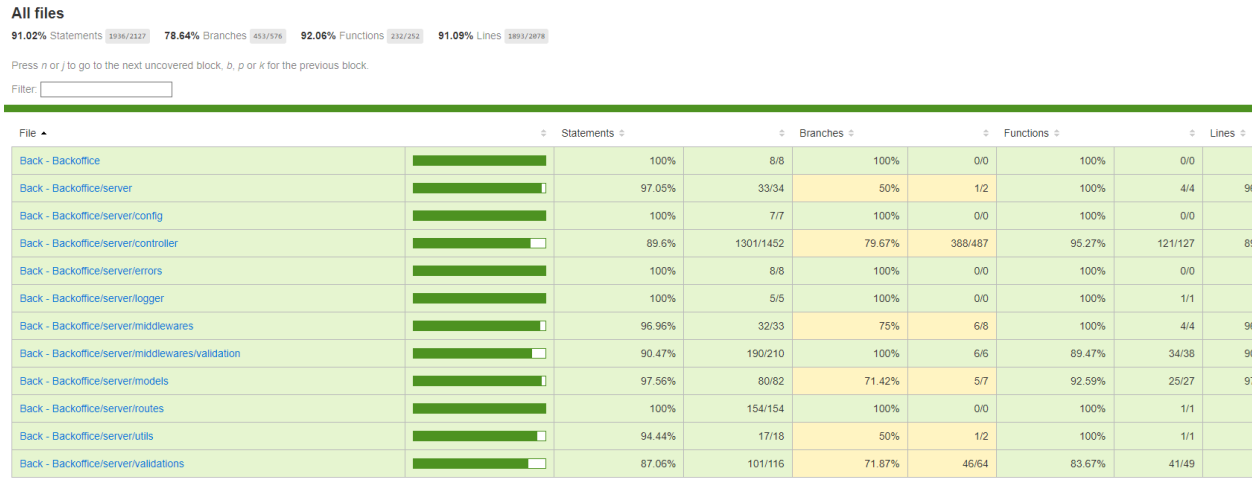
```
  },
  {
    "id": "632638ec86af6009dd818ea4",
    "amount": 100000,
    "term": 3,
    "rate": 0.15,
    "userId": "63262dd9f944867773f4638a",
    "targetSchemaId": "6321162b88d6e09094f568ba",
    "targetName": "NEW",
    "startDate": "2022-02-01T00:00:00Z",
    "endDate": "2022-04-30T00:00:00Z",
    "quota": 34170.12,
    "debt": 102510.36,
    "loanHistory": [
      {
        "monthStart": "2022-02-01T00:00:00Z",
        "monthEnd": "2022-02-28T00:00:00Z",
        "accumulated": 0,
        "monthDebt": 34170.12,
        "paymentId": "000000000000000000000000"
      },
      {
        "monthStart": "2022-03-01T00:00:00Z",
        "monthEnd": "2022-03-31T00:00:00Z",
        "accumulated": 0,
        "monthDebt": 68340.24,
        "paymentId": "000000000000000000000000"
      },
```

Pruebas unitarias:

Por tiempo no alcance a hacer muchas pruebas unitarias, y las pruebas unitarias que se realizaron no crean mocks o hacen stubs por lo que estás al correrse están haciendo llamados a la base de datos, yo se que esto no debería pasar jaja, pero por tiempo no me quedo de otra, por lo que para correr las pruebas unitarias es necesario montar el docker compose up, y correr el comando del readme en otra terminal.

```
PS C:\Users\juanp\Documents\Develop\Go\payment-api> go test ./tests/...
ok      github.com/jpbmdev/payment-api/tests/loan       (cached)
?       github.com/jpbmdev/payment-api/tests/mocks      [no test files]
ok      github.com/jpbmdev/payment-api/tests/targetSchema       (cached)
ok      github.com/jpbmdev/payment-api/tests/user       (cached)
```

Desafortunadamente tampoco me dio tiempo de implementar una librería para generar el archivo html de coverage, pero quisiera decir que yo si he trabajo generando coverages después de correr las pruebas unitarias:



All files

91.02% Statements 1936/2127 **78.64%** Branches 453/576 **92.06%** Functions 232/252 **91.09%** Lines 1893/2078

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
Back - Backoffice	100%	8/8	100%	0/0	100%	0/0		
Back - Backoffice/server	97.05%	33/34	50%	1/2	100%	4/4	9	
Back - Backoffice/server/config	100%	7/7	100%	0/0	100%	0/0		
Back - Backoffice/server/controller	89.6%	1301/1452	79.67%	388/487	95.27%	121/127	8	
Back - Backoffice/server/errors	100%	8/8	100%	0/0	100%	0/0		
Back - Backoffice/server/logger	100%	5/5	100%	0/0	100%	1/1		
Back - Backoffice/server/middlewares	96.96%	32/33	75%	6/8	100%	4/4	9	
Back - Backoffice/server/middlewares/validation	90.47%	190/210	100%	6/6	89.47%	34/38	9	
Back - Backoffice/server/models	97.56%	80/82	71.42%	5/7	92.59%	25/27	9	
Back - Backoffice/server/routes	100%	154/154	100%	0/0	100%	1/1		
Back - Backoffice/server/utils	94.44%	17/18	50%	1/2	100%	1/1		
Back - Backoffice/server/validations	87.06%	101/116	71.87%	46/64	83.67%	41/49		

All files / Back - Backoffice/server/validations common.js

87.06% Statements 101/116 **71.87%** Branches 46/64 **83.67%** Functions 41/49 **90%** Lines 90/100

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1     const {
2       body,
3       query,
4       validationResult,
5       check,
6       param,
7   1x  } = require("express-validator");
8   1x  const errors = require("../errors/");
9
10  1x  const existsInBody = (param) =>
11  511x  body(param).exists().withMessage(`Body must contain ${param}`);
12
13  165x  const optionalInBody = (param) => body(param).optional();
14
15  1x  const existsInQuery = (param) =>
16        query(param).exists().withMessage(`Query params must contain ${param}`);
17
18  167x  const optionalInQuery = (param) => query(param).optional();
19
20  1x  const existsInParam = (value) =>
```

Adjunto fotos de coverages de otros proyectos ajenos, Muchas gracias.