

# Contribuição de teste de Software Livre no framework de microserviços Moleculer

Rafael R. Soratto<sup>1</sup>

<sup>1</sup>Universidade Tecnológica Federal do Paraná  
Rua Rosalina Maria Ferreira, 1233 - Vila Carolo,  
Campo Mourão - PR, 87301-899

soratto@alunos.utfpr.edu.br

**Abstract.** *The objective of this work is to make a contribution to free software projects regarding the implementation and maintenance of tests. The contribution will be made through a Pull Request containing unit test cases generated from test requirements derived from test criteria. For this, we chose the Moleculer project (implemented with nodejs) and presented an architectural report of this open source software system (github).*

**Resumo.** *O objetivo deste trabalho é realizar uma contribuição para projetos de software livre referente a implementação e manutenção de testes. A contribuição será feita por meio de um Pull Request contendo casos de teste de unidade gerados a partir de requisitos de teste derivados de critérios de teste. Para isto, escolhemos o projeto Moleculer (implementado com nodejs) e apresentamos um relatório arquitetônico deste sistema de software com código aberto (github).*

---

## Contents

<b>1 A escolha do projeto Molecular</b>	<b>3</b>
1.1 Clone, Instalação das dependências e Execução dos testes Projeto	3
1.2 Como contribuir para o Molecular . . . . .	4
<b>2 Arquitetura e Implementação do Sistema</b>	<b>4</b>
2.1 Implementação dos Casos de Teste . . . . .	5
2.2 Cobertura dos casos de teste . . . . .	6
<b>3 Contribuição para os casos de teste</b>	<b>6</b>
<b>4 Nossa contribuição (PR)</b>	<b>10</b>
4.1 Identificando falha intermitente . . . . .	10
4.2 Solução proposta . . . . .	12
4.3 Revisão dos mantenedores . . . . .	17
<b>5 Conclusões</b>	<b>19</b>
<b>6 Automatizando o Processo de Identificação com Shaker</b>	<b>21</b>
6.1 Execução Local . . . . .	21
6.2 Execução no Github Actions . . . . .	21

---

## 1. A escolha do projeto **Molecular**

A intenção do projeto é contribuir com projetos de software livre com testes boa qualidade e, ao mesmo tempo, responder as seguintes questões de pesquisa:

1. Como é a cobertura dos atuais testes em um projeto de software livre ?
2. Como cobertura e qualidade dos testes podem ser melhoradas por meio do desenvolvimento de casos de teste derivados a partir de critérios de testes clássicos?
3. Como as contribuições que contém apenas casos de teste é aceita pelas comunidades ?
4. Quais as dificuldades de escrever casos de teste no mundo real?

O primeiro passo foi definir uma linguagem onde existe uma comunidade ativa contribuindo diariamente para projetos de código livre. No presente estado da arte grande parte da web utiliza o Javascript e por consequência o framework Node.js. Por este motivo iniciamos a busca em projetos que estão voltados para web implementados com Node.js.

Para especificar o escopo de projetos a serem analisados foi definido o seguinte critério de inclusão: são analisados somente os projetos de código livre que são utilizados para implementação de sistemas distribuídos (por exemplo, micro-serviços). Identificamos um projeto nesta área chamado **“Molecular”**: um framework rápido e moderno para micro-serviços implementados em Node.js. Esta biblioteca ajuda os desenvolvedores criarem serviços escaláveis, eficientes e re-utilizáveis <sup>1</sup>. Esse sistema é implementado em Node.js e pode ser utilizado na versão 16.18.0 do npm. Uma característica que definiu a escolha do projeto é que ele possui uma grande quantidade de testes automáticos implementados e eles correspondem a uma boa cobertura do código fonte. Outro ponto que favoreceu a escolha foi a recomendação de um colega de trabalho para utilizar este framework para desenvolver micro-serviços; mesmo sendo relativamente novo, possui uma comunidade que contribui diariamente para seu desenvolvimento.

Após realizar o clone e a execução do projeto em uma máquina local, foram identificadas 107 classes nesta biblioteca, e um total de 2425 casos de testes divididos em 140 suítes de teste.

### 1.1. Clone, Instalação das dependências e Execução dos testes Projeto

1. `git clone https://github.com/molecularjs/molecular`
2. `cd molecular.`
3. `npm i # OR yarn;`
4. `npm run test # or yarn test`

---

<sup>1</sup><https://github.com/molecularjs/molecular>

---

## 1.2. Como contribuir para o Moleculer

Existe uma comunidade engajada de contribuidores neste projeto. As contribuições são bem-vindas em diversos níveis do projeto, são eles: Contribuições de implementação de funcionalidades e solução de problemas. Para facilitar este processo é fornecido um guia de contribuição <sup>2</sup>.

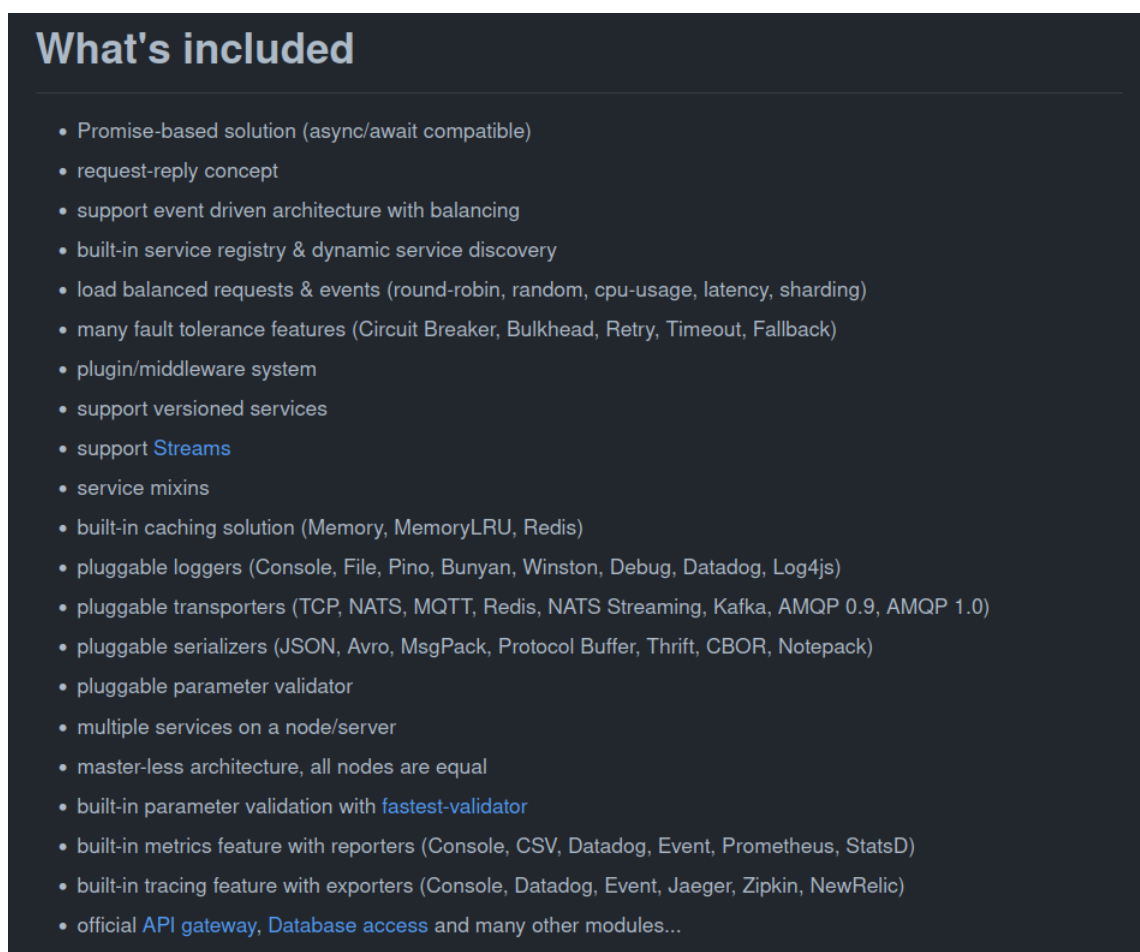
Neste documento, são apresentadas boas práticas para escrita do código fonte:

1. Use tabs with size of 4 for indents.
2. Always use strict mode & semicolons.
3. Use double quotes instead of single quotes.

O projeto Moleculer possui um FAQ e um chat no Discorda além de estar aberto para a criação de novas issues.

## 2. Arquitetura e Implementação do Sistema

As “**features**” presentes no Moleculer estão na Figura 1.



**Figure 1. Features do software Moleculer**

---

<sup>2</sup><https://github.com/moleculerjs/moleculer/blob/master/CONTRIBUTING.md>

Toda a implementação do Moleculer é feita na pasta 'src'. A classe principal do projeto é chamada de MoleculerRunner e está presente no arquivo 'runner.js' <sup>3</sup>. Todas as outras funções e classes são derivadas deste desta classe principal. Os atributos desta classe são:

```
1  class MoleculerRunner {
2  constructor() {
3    this.watchFolders = [];
4
5    this.flags = null;
6    this.configFile = null;
7    this.config = null;
8    this.servicePaths = null;
9    this.broker = null;
10   this.worker = null;
11  }
12  ...methods,
13 }
```

## 2.1. Implementação dos Casos de Teste

O projeto Moleculer utiliza o framework jest para execução dos testes. As seguintes bibliotecas fornecem o ambiente de execução dos testes:

```
1  "jest": "^27.5.1",
2  "jest-cli": "^27.5.1",
3  "jest-diff": "^27.5.1",
```

Essa informação de versões de bibliotecas estão disponíveis no arquivo "package.json", neste arquivo também se faz presente a configuração do jest:

```
1  "jest": {
2    "coverageDirectory": "../coverage",
3    "coveragePathIgnorePatterns": [
4      "/node_modules/",
5      "/test/services/",
6      "/test/typescript/",
7      "/test/unit/utils.js",
8      "/src/serializers/proto/",
9      "/src/serializers/thrift/"
10   ],
11   "transform": {},
12   "testEnvironment": "node",
13   "rootDir": "./src",
14   "roots": [
```

<sup>3</sup><https://github.com/moleculerjs/moleculer/blob/master/src/runner.js>

```

15     "../test"
16   ]
17 }

```

## 2.2. Cobertura dos casos de teste

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	96.62	88.41	94.42	97.02	
src	96.44	91.21	97.27	96.55	
async-storage.js	100	100	100	100	
constants.js	100	100	100	100	
context.js	98.87	89.51	100	99.35	167
cpu-usage.js	100	100	100	100	
errors.js	98.91	95.38	100	98.87	38
health.js	96.66	50	100	96.66	49
internals.js	100	100	100	100	
lock.js	100	83.33	100	100	32
logger-factory.js	85.48	90.62	86.66	86.27	88-95
middleware.js	100	100	100	100	
packets.js	100	100	100	100	
service-broker.js	95.97	89.32	96.09	96.13	178,336-340,603-610,735,1077,1164-1165,1632,1763-1764,1769-1770,1774-1777
service.js	100	97.16	100	100	56,466,487,768-785
transit.js	100	93.92	100	100	166,326,399,498,611,761,839-848,965,1105,1160-1175
utils.js	85.65	81.09	91.11	85.18	371-376,425-430,484,514,540-598
src/cachers	95.99	91.36	86.02	95.96	
base.js	99.16	96.34	78.94	99.11	463
index.js	100	100	100	100	
memory-lru.js	98.63	92.85	91.3	98.55	238
memory.js	100	89.18	95.65	100	197-208
redis.js	89.3	79.59	83.33	89.47	79,144-152,237-238,376-385,412-413,425,448-449
src/loggers	99.43	94.53	97.5	99.67	
base.js	100	85.71	100	100	58,64
bunyan.js	100	100	100	100	
console.js	95.45	91.66	100	100	35
datadog.js	100	100	93.33	100	
debug.js	100	100	100	100	
file.js	100	100	88.88	100	
formatted.js	100	90	100	100	132-135,167,228
index.js	100	100	100	100	
log4js.js	95.83	90	100	95	64
pino.js	100	100	100	100	
winston.js	100	93.33	100	100	75
src/metrics	92.28	90.9	86.04	91.66	
commons.js	85.54	55	60	85.45	419-438,554-558,567-596
constants.js	100	100	100	100	
index.js	100	100	100	100	
rates.js	100	100	100	100	
registry.js	100	98.01	100	100	308,323
src/metrics/reporters	95.14	77.45	97.97	97.63	
base.js	96.42	95	88.88	100	38
console.js	92.53	64	100	93.33	107,132,176,204
csv.js	97.7	84.37	100	100	108-126,152-172
datadog.js	94.36	66.66	94.44	98.43	291
event.js	96.15	90	100	100	69
index.js	95.23	92.85	100	94.11	39

Figure 2. Cobertura casos de teste

## 3. Contribuição para os casos de teste

A primeira dúvida levantada pelo professor é como definir uma classe que precisa ser testada seguindo abordagens mais rigorosas. A primeira idéia que surge é analisar a cobertura de código fonte, e implementar casos de testes para as classes que não possuem boa cobertura. Porém, ter uma boa cobertura nem sempre significa que o caso de teste cubra todos os fluxos que podem ser seguidos pelo software. Para provar isto realizamos a reexecução dos 2425 casos de teste 1000 vezes, para verificar se alguma falha intermitente ocorre em pelo menos uma execução.

Para armazenar os resultados das execuções utilizamos a biblioca do npm 'jest-junit' que salva os resultados das execuções dos casos de teste em

index.js	100	100	100	100	
node-catalog.js	98.71	90	92.85	98.63	105
node.js	100	90	85.71	100	87,109
registry.js	99.39	91.01	100	99.33	317
service-catalog.js	97.22	92.85	100	100	121,148,178
service-item.js	100	100	100	100	
src/registry/discoverers	95.32	92.17	94.02	95.15	
base.js	98.93	94.44	88.46	98.8	227
etcd3.js	91.79	86.36	95.55	91.4	169-177,233,349-357
index.js	100	95	100	100	47
local.js	100	100	100	100	
redis.js	94.93	92.15	94.33	95.27	101,139-141,300,440,447
src/serializers	99.58	91	94.87	100	
avro.js	100	100	100	100	
base.js	98.76	90.38	75	100	78-79,125-126,186
cbor.js	100	100	100	100	
index.js	100	100	100	100	
json.js	100	100	100	100	
msgpack.js	100	100	100	100	
notepack.js	100	100	100	100	
protobuf.js	100	87.5	100	100	85-107
thrift.js	100	87.5	100	100	92-116
src/strategies	99.49	95.5	92.3	100	
base.js	100	100	50	100	
cpu-usage.js	100	100	100	100	
index.js	100	100	100	100	
latency.js	98.68	93.1	89.47	100	208-212
random.js	100	100	100	100	
round-robin.js	100	100	100	100	
shard.js	100	92.3	100	100	90,107
src/tracing	99.24	96.55	96.29	99.16	
index.js	100	100	100	100	
rate-limiter.js	100	100	100	100	
span.js	100	93.93	100	100	130-147
tracer.js	98.43	97.87	93.33	98.18	284
src/tracing/exporters	96.62	87.58	94.28	97.55	
base.js	100	100	62.5	100	
console.js	95.96	82.35	100	98.16	242-243
datadog.js	93.18	78.33	91.66	96.15	86,101-109
event-legacy.js	100	84.61	100	100	90-95,124
event.js	100	100	100	100	
index.js	95.45	92.85	100	94.44	41
jaeger.js	98.61	95.23	100	98.52	101
newrelic.js	95.23	88.46	92.3	94.87	119,129
zipkin.js	95.23	88.46	92.3	94.87	119,129
src/transporters	93.98	80.5	89.24	94.87	
amqp.js	90.83	81.92	83.87	92.24	221-222,229-230,263,336-342
amqp10.js	86.85	62.37	77.27	88.19	131-133,229-238,293-295,355,410,482-485,529-532,576-579
base.js	93.67	87.8	77.77	96	120-121,236
fake.js	91.66	100	77.77	90.9	89-100
index.js	97.5	97.5	100	100	53
kafka.js	100	75	100	100	143-148
mqtt.js	100	83.33	100	100	115-133
nats.js	98.94	91.17	100	98.86	59
redis.js	100	83.33	100	100	113-118
stan.js	100	83.33	100	100	40-124
tcp.js	94	81.94	92.15	94.75	377,383,400,480,484,508,592-608,625,685-686
src/transporters/tcp	96.46	80	98.38	96.96	
constants.js	100	100	100	100	
parser.js	100	100	100	100	
tcp-reader.js	100	62.5	100	100	52-135
tcp-writer.js	100	89.65	100	100	86-98
udp-broadcaster.js	91.07	70.45	95.65	92.59	139,141,151,159-160,233-236
src/validators	95.71	91.66	86.36	95.23	
base.js	91.42	88.88	78.57	90.9	29-50
fastest.js	100	100	100	100	
index.js	100	93.75	100	100	41

Test Suites: 140 passed, 140 total  
Tests: 4 skipped, 2421 passed, 2425 total  
Snapshots: 33 passed, 33 total  
Time: 41.799 s  
Ran all test suites.  
Force exiting Jest: Have you considered using `--detectOpenHandles` to detect async operations that kept running after all tests finished?  
Done in 42.86s.

Figure 3. Cobertura casos de teste

arquivos ‘.xml’. Após salvar os 1000 resultados em arquivos xml, criamos um script em Python para analisar todos arquivos e procurar pela tag ‘failure’ em cada xml. Desta forma conseguimos executar os casos de testes diversas vezes e analisar quais falhas ocorreram em cada execução. Após essas 1000 execuções, identificamos 3 falhas intermitentes.

A primeira foi identificada no seguinte caso de teste<sup>4</sup> :

```
1 [
2   {
3     "classname": "Test MsgPackSerializer-should serialize the
4     event packet",
5     "content": "Error: expect(received).toBe(expected) // Object
6     .is equality\n\nExpected: 144\nReceived: 140\n    at Object.<
7     anonymous> (/home/soratto/Documentos/workspace/moleculer/test/
8     unit/serializers/msgpack.spec.js:36:20)\n    at Promise.then.
9     completed (/home/soratto/Documentos/workspace/moleculer/
    node_modules/jest-circus/build/utils.js:391:28)\n    at new
    Promise (<anonymous>)\n    at callAsyncCircusFn (/home/soratto
    /Documentos/workspace/moleculer/node_modules/jest-circus/build
    /utils.js:316:10)\n    at _callCircusTest (/home/soratto/
    Documentos/workspace/moleculer/node_modules/jest-circus/build/
    run.js:218:40)\n    at processTicksAndRejections (node:
    internal/process/task_queues:96:5)\n    at _runTest (/home/
    soratto/Documentos/workspace/moleculer/node_modules/jest-
    circus/build/run.js:155:3)\n    at _runTestsForDescribeBlock
    (/home/soratto/Documentos/workspace/moleculer/node_modules/
    jest-circus/build/run.js:66:9)\n    at
    _runTestsForDescribeBlock (/home/soratto/Documentos/workspace/
    moleculer/node_modules/jest-circus/build/run.js:60:9)\n    at
    run (/home/soratto/Documentos/workspace/moleculer/node_modules
    /jest-circus/build/run.js:25:3)",
    "filename": "junit.xml-6d507640-5d2b-11ed-9f49-577817ba0a3b.
    xml",
    "file_test_name": "test/unit/serializers/msgpack.spec.js",
    "count": 1
  },
]
```

A segunda foi identificada no seguinte caso de teste<sup>5</sup> :

```
1 {
2   "classname": "Test NotePackSerializer-should serialize the
3   event packet",
4   "content": "Error: expect(received).toBe(expected) // Object
5   .is equality\n\nExpected: 144\nReceived: 140\n    at Object.<
6   anonymous> (/home/soratto/Documentos/workspace/moleculer/test/
```

<sup>4</sup><https://github.com/moleculerjs/moleculer/blob/master/test/unit/serializers/msgpack.spec.js>

<sup>5</sup><https://github.com/moleculerjs/moleculer/blob/master/test/unit/serializers/msgpack.spec.js>



```

unit/serializers/notepack.spec.js:36:20)\n    at Promise.then.
completed (/home/soratto/Documentos/workspace/moleculer/
node_modules/jest-circus/build/utils.js:391:28)\n    at new
Promise (<anonymous>)\n    at callAsyncCircusFn (/home/soratto
/Documentos/workspace/moleculer/node_modules/jest-circus/build
/utils.js:316:10)\n    at _callCircusTest (/home/soratto/
Documentos/workspace/moleculer/node_modules/jest-circus/build/
run.js:218:40)\n    at processTicksAndRejections (node:
internal/process/task_queues:96:5)\n    at _runTest (/home/
soratto/Documentos/workspace/moleculer/node_modules/jest-
circus/build/run.js:155:3)\n    at _runTestsForDescribeBlock
(/home/soratto/Documentos/workspace/moleculer/node_modules/
jest-circus/build/run.js:66:9)\n    at
_runTestsForDescribeBlock (/home/soratto/Documentos/workspace/
moleculer/node_modules/jest-circus/build/run.js:60:9)\n    at
run (/home/soratto/Documentos/workspace/moleculer/node_modules
/jest-circus/build/run.js:25:3)",
4     "filename": "junit.xml-bab47a70-5d45-11ed-9b60-ed1ef21011f9.
xml",
5     "file_test_name": "test/unit/serializers/notepack.spec.js",
6     "count": 1
7   }

```

A terceira foi identificada no seguinte caso de teste<sup>6</sup> :

```

1  {
2    "classname": "Test CborSerializer-should serialize the event
packet",
3    "content": "Error: expect(received).toBe(expected) // Object
.is equality\n\nExpected: 150\nReceived: 146\n    at Object.<
anonymous> (/home/soratto/Documentos/workspace/moleculer/test/
unit/serializers/cbor.spec.js:36:20)\n    at Promise.then.
completed (/home/soratto/Documentos/workspace/moleculer/
node_modules/jest-circus/build/utils.js:391:28)\n    at new
Promise (<anonymous>)\n    at callAsyncCircusFn (/home/soratto
/Documentos/workspace/moleculer/node_modules/jest-circus/build
/utils.js:316:10)\n    at _callCircusTest (/home/soratto/
Documentos/workspace/moleculer/node_modules/jest-circus/build/
run.js:218:40)\n    at processTicksAndRejections (node:
internal/process/task_queues:96:5)\n    at _runTest (/home/
soratto/Documentos/workspace/moleculer/node_modules/jest-
circus/build/run.js:155:3)\n    at _runTestsForDescribeBlock

```

<sup>6</sup><https://github.com/moleculerjs/moleculer/blob/master/test/unit/serializers/msgpack.spec.js>

```

    (/home/soratto/Documentos/workspace/moleculer/node_modules/
    jest-circus/build/run.js:66:9)\n    at
    _runTestsForDescribeBlock (/home/soratto/Documentos/workspace/
    moleculer/node_modules/jest-circus/build/run.js:60:9)\n    at
    run (/home/soratto/Documentos/workspace/moleculer/node_modules
    /jest-circus/build/run.js:25:3)",
4     "filename":"junit.xml-24982560-5d49-11ed-89a0-e11440f676bf.
    xml",
5     "file_test_name":"test/unit/serializers/cbor.spec.js",
6     "count":1
7 }

```

O mais interessante foi o fato de que esses erros foram encontrados em classes que de acordo com a coverage da suíte, possuíam 100% de suas linhas testadas. Isso significa que a abordagem ‘todos nós’ pode não encontrar todos erros que abordagens ‘todas arestas’ ou mistas podem descobrir.

## 4. Nossa contribuição (PR)

Conforme apresentado na seção anterior os casos de teste possuem falhas intermitentes nas classes `MsgPackSerializer`, `CborSerializer` e `NotePackSerializer`. Ambas são classes responsáveis pela serialização de um objeto. A falha ocorre ao acessar o atributo tamanho de um vetor. Em todas falhas o erro ocorreu porque o vetor tinha 4 posições a menos do que o esperado. Sempre que isso ocorre, os testes falham.

### 4.1. Identificando falha intermitente

Para verificar qual problema ocorre, analisamos o CUT (code under the test) com a finalidade de verificar quais variáveis revelam a falha intermitente. O conteúdo do caso de teste sobre serialização é o seguinte:

```

1 describe("Test MsgPackSerializer", () => {
2   let serializer = new MsgPackSerializer();
3   serializer.init();
4
5   it("should serialize the event packet", () => {
6     const now = new Date("2022-11-06T22:59:47.000Z");
7     const obj = {
8       ver: "4",
9       sender: "node-100",
10      id: "8b3c7371-7f0a-4aa2-b734-70ede29e1bbb",
11      event: "user.created",
12      data: {
13        a: 5,
14        b: "Test",
15        c: now
16      },

```

```

17     broadcast: true,
18     meta: {},
19     level: 1,
20     needAck: false
21   }
22
23   const s = serializer.serialize(cloneDeep(obj), P.PACKET_EVENT);
24
25   expect(s.length).toBe(144);
26
27   const res = serializer.deserialize(s, P.PACKET_EVENT);
28   expect(res).not.toBe(obj);
29   expect(res).toEqual(obj);
30   });
31 });

```

Neste caso de teste é possível visualizar a utilização do horário atual para preencher a variável 'c' presente no objeto 'data'. Surgiu a suspeita de que, como o horário atual varia, ele pode ser a causa raiz da instabilidade. Para verificar isto executamos o teste da classe MsgPackSerializer mais 1000 vezes, e quando ocorreu essa diferença de tamanho de 144 para 140 salvamos o objeto utilizado para teste. Portanto, o objeto com resultado instável é o seguinte:

```

1  {
2    ver: "4",
3    sender: "node-100",
4    id: "8b3c7371-7f0a-4aa2-b734-70ede29e1bbb",
5    event: "user.created",
6    data: {
7      a: 5,
8      b: "Test",
9      c: "2022-11-06T22:59:47.000Z"
10   },
11   broadcast: true,
12   meta: {},
13   level: 1,
14   needAck: false
15 }

```

Analisando o objeto que resulta em uma falha intermitente, ele possui um valor de data interessante: que termina com '000Z'. Para simular um teste que falha, definimos um **mutante do teste** que incluía este valor de data, e descobrimos que ele sempre falha:

```

1 describe("Test MsgPackSerializer", () => {
2   let serializer = new MsgPackSerializer();
3   serializer.init();
4

```

```

5  it("should serialize the event packet", () => {
6      const now = new Date("2022-11-06T22:58:47.000Z")
7      const obj = {
8          ver: "4",
9          sender: "node-100",
10         id: "8b3c7371-7f0a-4aa2-b734-70ede29e1bbb",
11         event: "user.created",
12         data: {
13             a: 5,
14             b: "Test",
15             c: now
16         },
17         broadcast: true,
18         meta: {},
19         level: 1,
20         needAck: false
21     }
22
23     const s = serializer.serialize(cloneDeep(obj), P.PACKET_EVENT);
24
25     expect(s.length).toBe(154);
26
27     const res = serializer.deserialize(s, P.PACKET_EVENT);
28     expect(res).not.toBe(obj);
29     expect(res).toEqual(obj);
30 });
31 });

```

Conclui-se que o mutante que revela a falha são aqueles que possuem uma data que termine com '000Z'.

Para detalhar este problema foi escrita uma issue no projeto no seguinte endereço: <https://github.com/moleculerjs/moleculer/issues/1150>.

## 4.2. Solução proposta

Para solucionar este problema nas classes escrevemos o seguinte PR (Pull Request): <https://github.com/moleculerjs/moleculer/pull/1151>.

A seguinte solução foi proposta: implementar casos de testes que comprovem que os objetos que possuem o atributo de data podem ser instáveis ao serem serializados, e utilizar a função 'toUTCString()' no objeto antes da serialização para tornar a verificação do teste estável e não variar o número de bytes verificado pelo teste.

Um dos algoritmos de serialização utilizados é o msgpack. Em sua documentação <sup>7</sup> podemos notar que para datas que utilizam microssegundos é necessário um espaço maior em bytes para o armazenamento:

<sup>7</sup><https://github.com/msgpack/msgpack/blob/master/spec.md#timestamp-extension-type>



**Figure 4. Valor em bytes para armazenar timestamp no msgpack**

Portanto, utilizamos a técnica de mutação do teste: o mutante gerado possui uma data fixada com o valor 000z em seu fim. Este mutante foi encontrado após 1000 reexecuções do caso de teste. Para solucionar ele, criamos 4 casos de testes novos:

1. Um caso de teste para quando se utiliza a serialização de um objeto com a data que termine com 000Z.
  - O resultado esperado é um objeto serializado com 4 bytes a menos do que o esperado anteriormente pelo teste;
2. Um caso de teste para quando se utiliza a serialização de um objeto com a data que termine com 001Z.
  - O resultado esperado é um objeto serializado com a quantidade de bytes a idêntica ao esperado anteriormente pelo teste;
3. Um caso de teste para quando se utiliza a serialização de um objeto com a data que termine com 001Z utilizando a função `.toUTCString()`.
  - O resultado esperado é um objeto serializado com a quantidade de bytes estável.
4. Um caso de teste para quando se utiliza a serialização de um objeto com a data que termine com 000Z utilizando a função `.toUTCString()`.
  - O resultado esperado é um objeto serializado com a quantidade de bytes estável.

Para cada uma das 3 classes adicionamos esses 4 casos de testes totalizando 12 casos de testes adicionados. Por exemplo, a classe `msgpack.spec.js` ficou da seguinte forma:

```
1 const MsgPackSerializer = require("../.../src/serializers/msgpack");
2 const { cloneDeep } = require("lodash");
3 const P = require("../.../src/packets");
4
5 describe("Test MsgPackSerializer constructor", () => {
6   it("should create an empty options", () => {
7     let serializer = new MsgPackSerializer();
8     expect(serializer).toBeDefined();
9     expect(serializer.serialize).toBeDefined();
10    expect(serializer.deserialize).toBeDefined();
11  });
12 });
13
14 describe("Test MsgPackSerializer", () => {
15   let serializer = new MsgPackSerializer();
16   serializer.init();
17
18   it("should serialize the event packet (with UTC String)", () => {
19     const now = new Date().toUTCString();
20     const obj = {
21       ver: "4",
22       sender: "node-100",
23       id: "8b3c7371-7f0a-4aa2-b734-70ede29e1bbb",
24       event: "user.created",
25       data: {
26         a: 5,
27         b: "Test",
28         c: now
29       },
30       broadcast: true,
31       meta: {},
32       level: 1,
33       needAck: false
34     };
35     const s = serializer.serialize(cloneDeep(obj), P.PACKET_EVENT);
36     expect(s.length).toBe(164);
37
38     const res = serializer.deserialize(s, P.PACKET_EVENT);
39     expect(res).not.toBe(obj);
40     expect(res).toEqual(obj);
41   });
42   it("should serialize the event packet (date time ends 000Z using UTC)", () => {
43     const now = new Date("2022-11-06T22:59:47.000Z").toUTCString();
44     const obj = {
45       ver: "4",
46       sender: "node-100",
47       id: "8b3c7371-7f0a-4aa2-b734-70ede29e1bbb",
48       event: "user.created",
49       data: {
50         a: 5,
51         b: "Test",
```

```

52         c: now
53     },
54     broadcast: true,
55     meta: {},
56     level: 1,
57     needAck: false
58 };
59
60 const s = serializer.serialize(cloneDeep(obj), P.PACKET_EVENT);
61 expect(s.length).toBe(164);
62
63 const res = serializer.deserialize(s, P.PACKET_EVENT);
64 expect(res).not.toBe(obj);
65 expect(res).toEqual(obj);
66 });
67
68 it("should serialize the event packet (date time ends with 001Z and using UTC
69 )", () => {
70     const now = new Date("2022-11-06T22:59:47.001Z").toUTCString();
71     const obj = {
72         ver: "4",
73         sender: "node-100",
74         id: "8b3c7371-7f0a-4aa2-b734-70ede29e1bbb",
75         event: "user.created",
76         data: {
77             a: 5,
78             b: "Test",
79             c: now
80         },
81         broadcast: true,
82         meta: {},
83         level: 1,
84         needAck: false
85     };
86
87     const s = serializer.serialize(cloneDeep(obj), P.PACKET_EVENT);
88     expect(s.length).toBe(164);
89
90     const res = serializer.deserialize(s, P.PACKET_EVENT);
91     expect(res).not.toBe(obj);
92     expect(res).toEqual(obj);
93 });
94
95 it("should serialize the event packet (date time ends with 000Z without UTC)"
96 , () => {
97     const now = new Date("2022-11-06T22:59:47.000Z");
98     const obj = {
99         ver: "4",
100         sender: "node-100",
101         id: "8b3c7371-7f0a-4aa2-b734-70ede29e1bbb",
102         event: "user.created",
103         data: {
104             a: 5,

```

```

103     b: "Test",
104     c: now
105   },
106   broadcast: true,
107   meta: {},
108   level: 1,
109   needAck: false
110 };
111
112 const s = serializer.serialize(cloneDeep(obj), P.PACKET_EVENT);
113 expect(s.length).toBe(140);
114
115 const res = serializer.deserialize(s, P.PACKET_EVENT);
116 expect(res).not.toBe(obj);
117 expect(res).toEqual(obj);
118 });
119
120 it("should serialize the event packet (date time ends with 001Z without UTC)"
121 , () => {
122   const now = new Date("2022-11-06T22:59:47.001Z");
123   const obj = {
124     ver: "4",
125     sender: "node-100",
126     id: "8b3c7371-7f0a-4aa2-b734-70ede29e1bbb",
127     event: "user.created",
128     data: {
129       a: 5,
130       b: "Test",
131       c: now
132     },
133     broadcast: true,
134     meta: {},
135     level: 1,
136     needAck: false
137   };
138
139   const s = serializer.serialize(cloneDeep(obj), P.PACKET_EVENT);
140   expect(s.length).toBe(144);
141
142   const res = serializer.deserialize(s, P.PACKET_EVENT);
143   expect(res).not.toBe(obj);
144   expect(res).toEqual(obj);
145 });

```

Porém, para chegar neste resultados, foram necessários diversos commits. Isto porque minha solução inicial não era a correta, e mesmo com a aprovação do mantenedor o pipeline revelou algumas falhas.



### 4.3. Revisão dos mantenedores

Procuramos facilitar o processo dos mantenedores e escrevemos uma issue e um PR bem detalhado conforme apresenta a Figura 5:

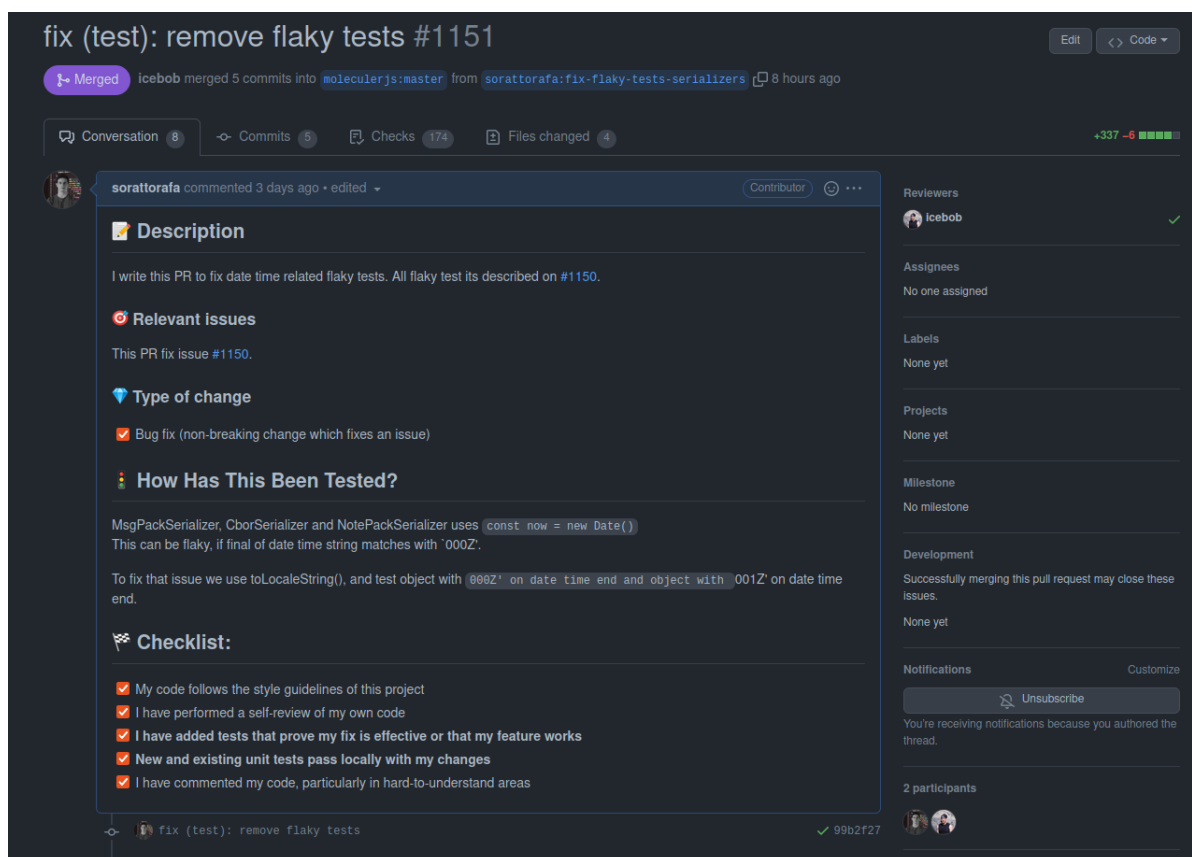
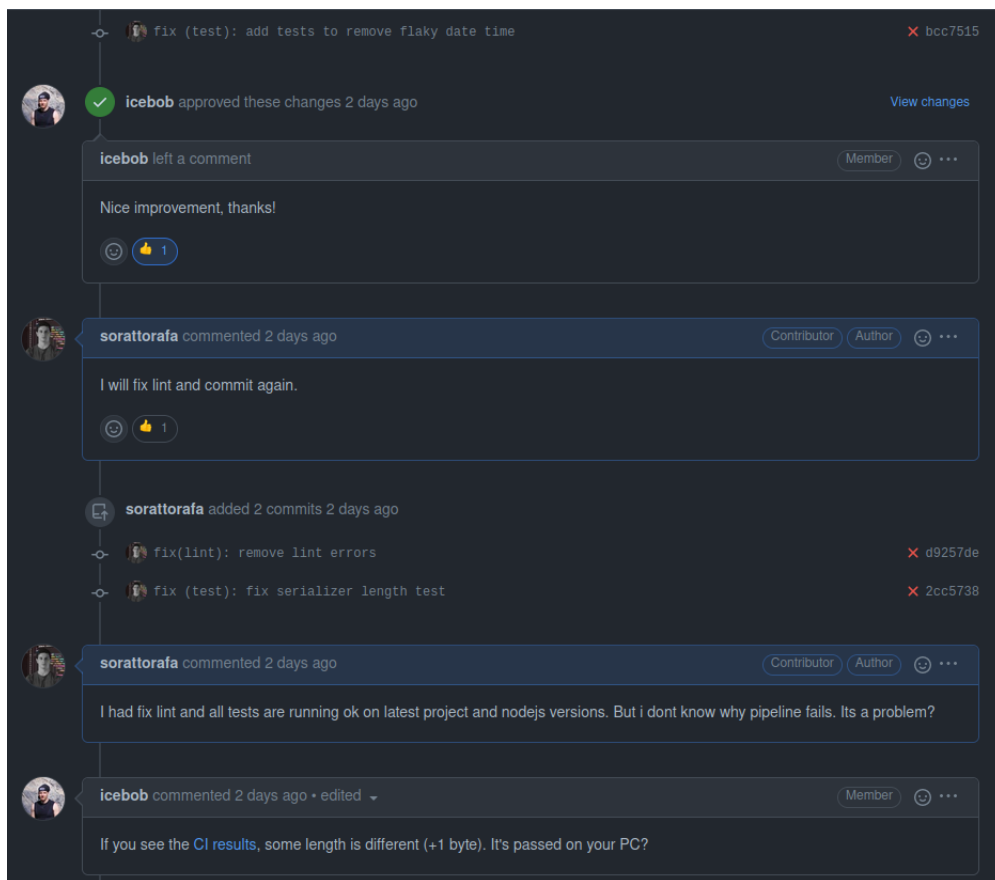


Figure 5. Criação do PR

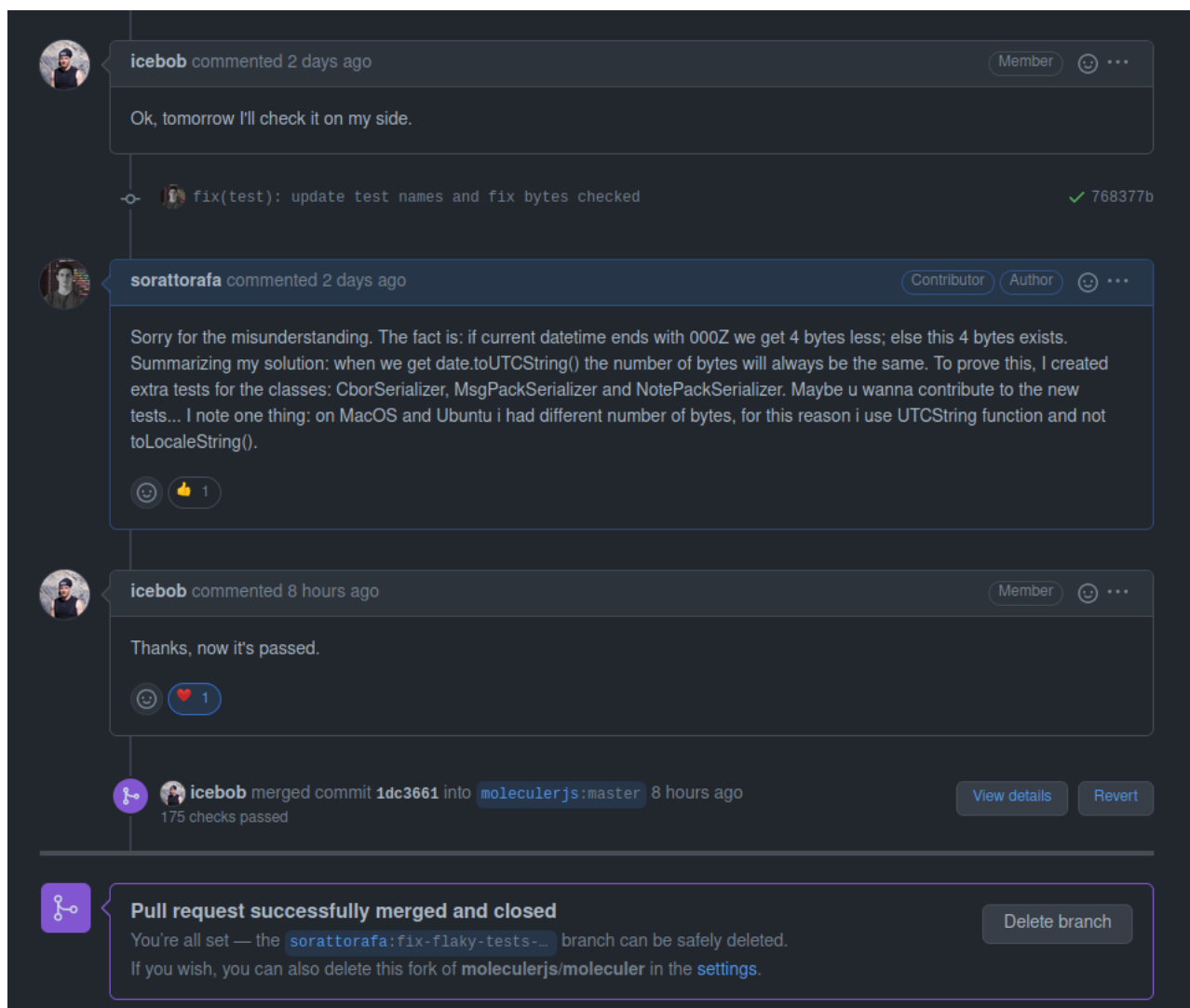
Após a criação do Pull Request recebemos o primeiro feedback em 1 dia, agradecendo a contribuição conforme apresentado na Figura 6:



**Figure 6. Feedback sobre o Pull Request**

Porém, mesmo com o problema identificado, a solução correta não tinha sido encontrada, e o pipeline falhou nos meus primeiros commits conforme apresenta a Figura 6.

Analisando o pipeline, consegui verificar e corrigir o erro. Após fazer isso, expliquei a situação pro mantenedor e conclui que a solução estava finalizada, conforme apresenta a Figura 7:



**Figure 7. Merged Pull Request**

## 5. Conclusões

1. Como é a cobertura dos atuais testes em um projeto de software livre ?
  - Resposta: No projeto molecular, a maioria dos arquivos possuem mais de 90% de cobertura por linhas de código. Tendo apenas uns 5 arquivos com a porcentagem em 85% no mínimo.
2. Como cobertura e qualidade dos testes podem ser melhoradas por meio do desenvolvimento de casos de teste derivados a partir de critérios de testes clássicos?
  - Resposta: Identificamos uma falha em um caso de teste que possuía 100% de cobertura criando um mutante do teste. Este mutante foi descoberto em uma situação de reexecução forçada, ou seja, executamos o teste até uma situação instável ser descoberta.
3. Como as contribuições que contêm apenas casos de teste é aceita pelas comunidades ?

- 
- Resposta: Como identificamos uma falha real do código a ser testado (CUT) tivemos um apoio rápido do dono do projeto. Como seguimos as diretrizes de contribuição tivemos poucos problemas com a comunidade. Nossa única falha ocorreu com o pipeline, mas após ser corrigida o pull request foi aceito.

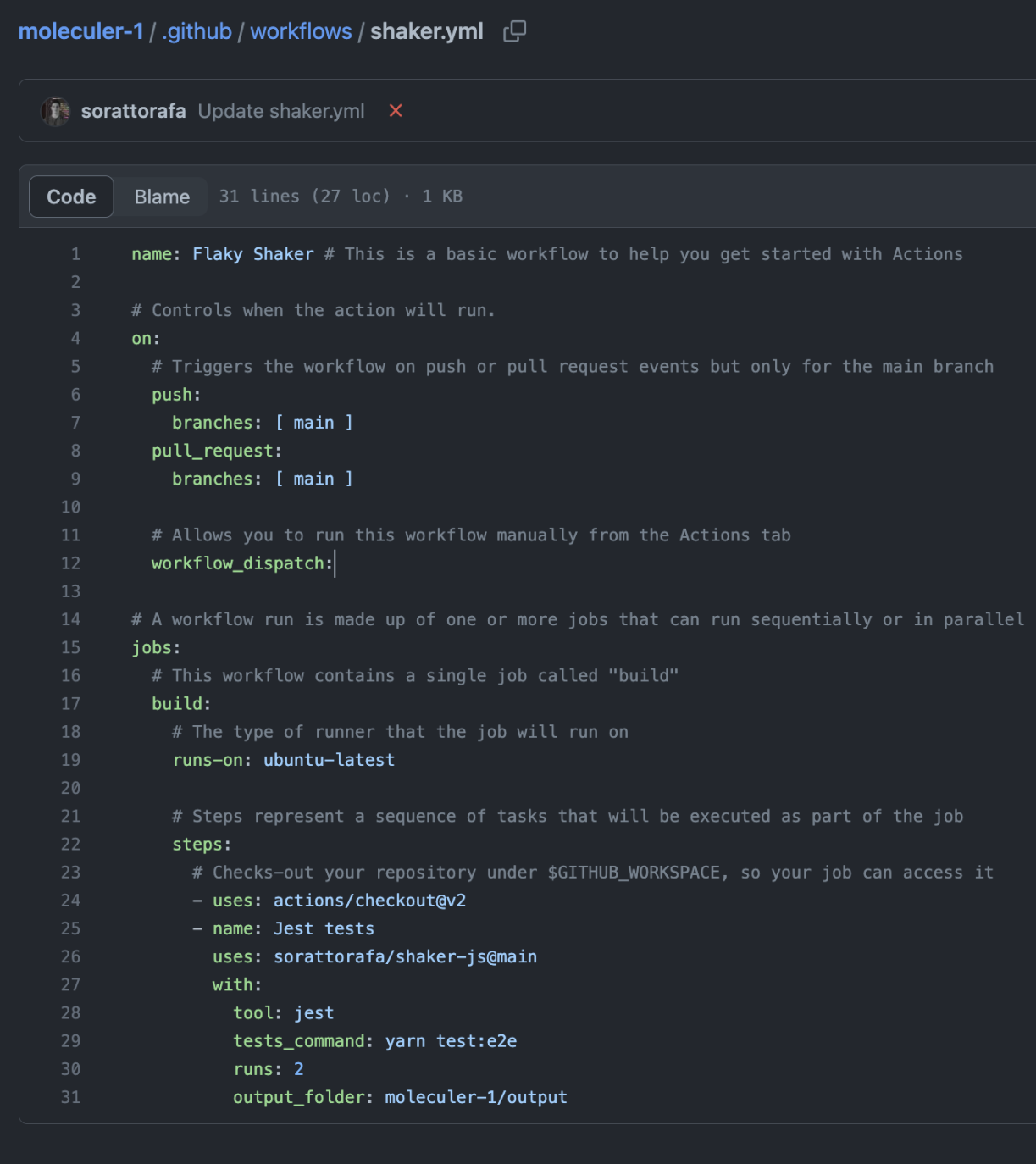
4. Quais as dificuldades de escrever casos de teste no mundo real? Resposta:

- (a) Identificar um projeto com casos de testes bem definidos;
- (b) Identificar um cenário de teste que não está bem estabelecido pelo software;
- (c) Identificar uma abordagem para cobrir o cenário que não foi testado;
- (d) Conseguir apresentar sua contribuição como relevante para o projeto.

## 6. Automatizando o Processo de Identificação com Shaker

### 6.1. Execução Local

### 6.2. Execução no Github Actions



The screenshot shows a GitHub repository interface for a file named `shaker.yml` in the `moleculer-1` repository. The file is 31 lines long, 27 lines of code, and 1 KB in size. The code is a GitHub Actions workflow configuration. It starts with a `name` field set to `Flaky Shaker` and a comment. The `on` field is configured to trigger the workflow on `push` and `pull_request` events for the `main` branch. There is a `workflow_dispatch` event for manual triggering. The `jobs` section contains a single job named `build`. This job runs on `ubuntu-latest` and consists of a `steps` section. The first step checks out the repository using `actions/checkout@v2`. The second step is named `Jest tests` and uses the `sorattorafa/shaker-js@main` action. This step is configured with `tool: jest`, `tests_command: yarn test:e2e`, `runs: 2`, and `output_folder: moleculer-1/output`.

```
1  name: Flaky Shaker # This is a basic workflow to help you get started with Actions
2
3  # Controls when the action will run.
4  on:
5    # Triggers the workflow on push or pull request events but only for the main branch
6    push:
7      branches: [ main ]
8    pull_request:
9      branches: [ main ]
10
11   # Allows you to run this workflow manually from the Actions tab
12   workflow_dispatch:|
13
14   # A workflow run is made up of one or more jobs that can run sequentially or in parallel
15   jobs:
16     # This workflow contains a single job called "build"
17     build:
18       # The type of runner that the job will run on
19       runs-on: ubuntu-latest
20
21       # Steps represent a sequence of tasks that will be executed as part of the job
22       steps:
23         # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
24         - uses: actions/checkout@v2
25         - name: Jest tests
26           uses: sorattorafa/shaker-js@main
27           with:
28             tool: jest
29             tests_command: yarn test:e2e
30             runs: 2
31             output_folder: moleculer-1/output
```

Figure 8. Arquivo de configuração do actions