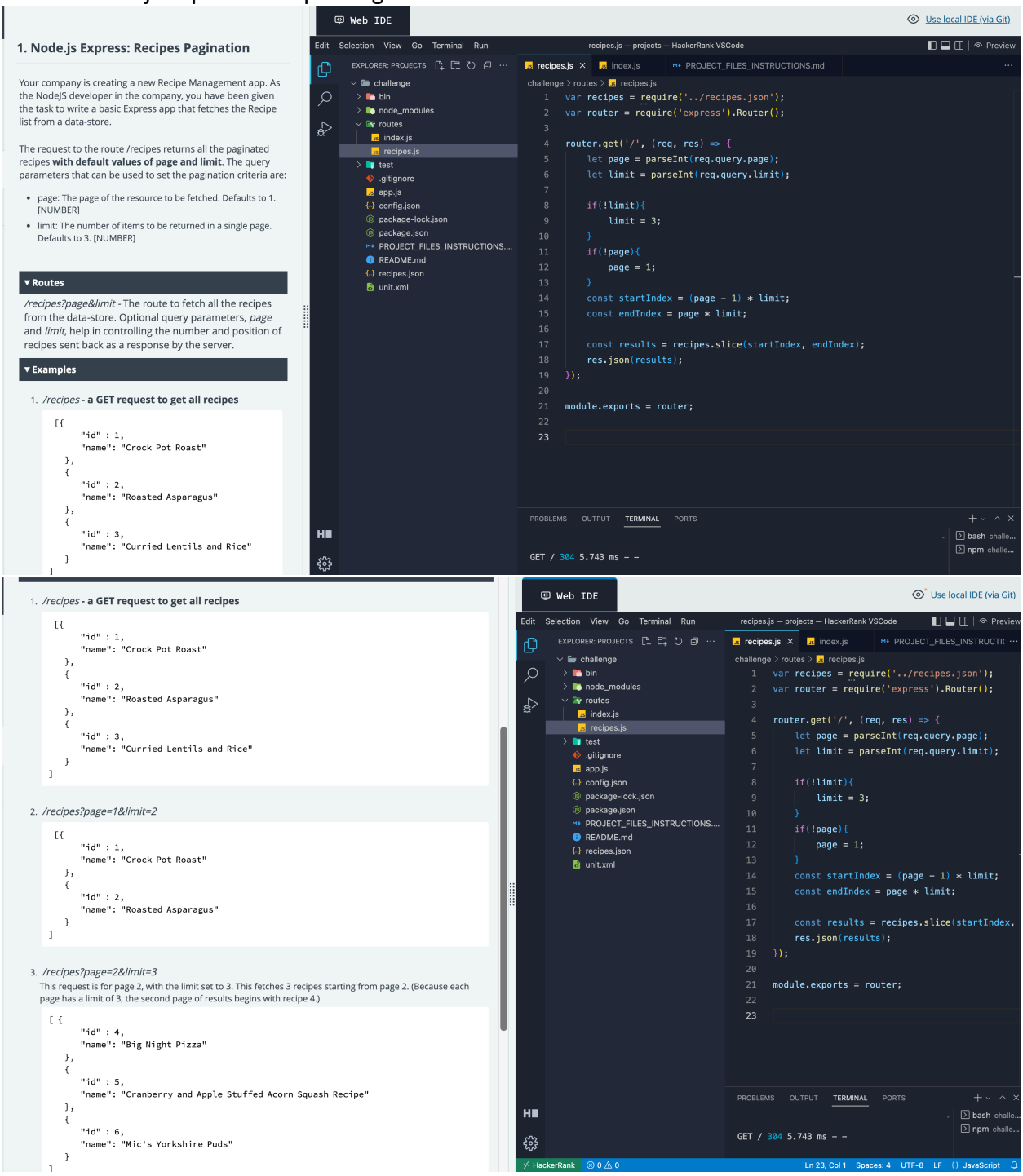
1. Node.js Express: Recipes Pagination



```
]
```

3. */recipes?page=2&limit=3*
   This request is for page 2, with the limit set to 3. This fetches 3 recipes starting from page 2. (Because each page has a limit of 3, the second page of results begins with recipe 4.)

```
[ {
      "id" : 4,
      "name": "Big Night Pizza"
   },
   {
      "id" : 5,
      "name": "Cranberry and Apple Stuffed Acorn Squash Recipe"
   },
   {
      "id" : 6,
      "name": "Mic's Yorkshire Puds"
   }
]
```

Software Instructions

The question(s) requires **Node 18.15.0 LTS or above**.

- Download & Install Node.JS

Git Instructions

Use the following commands to work with this project

Run Copy

```
npm start
```

Test Copy

```
npm test
```

Install Copy

```
npm install
```

1. Node.js Express: Recipes Pagination

Your company is creating a new Recipe Management app. As the NodeJS developer in the company, you have been given the task to write a basic Express app that fetches the Recipe list from a data-store.

The request to the route /recipes returns all the paginated recipes **with default values of page and limit**. The query parameters that can be used to set the pagination criteria are:

- page: The page of the resource to be fetched. Defaults to 1. [NUMBER]
- limit: The number of items to be returned in a single page. Defaults to 3. [NUMBER]

▸ **Routes**

▸ **Examples**

Software Instructions

The question(s) requires **Node 18.15.0 LTS or above**.

- Download & Install Node.JS

Git Instructions

Use the following commands to work with this project

Run Copy

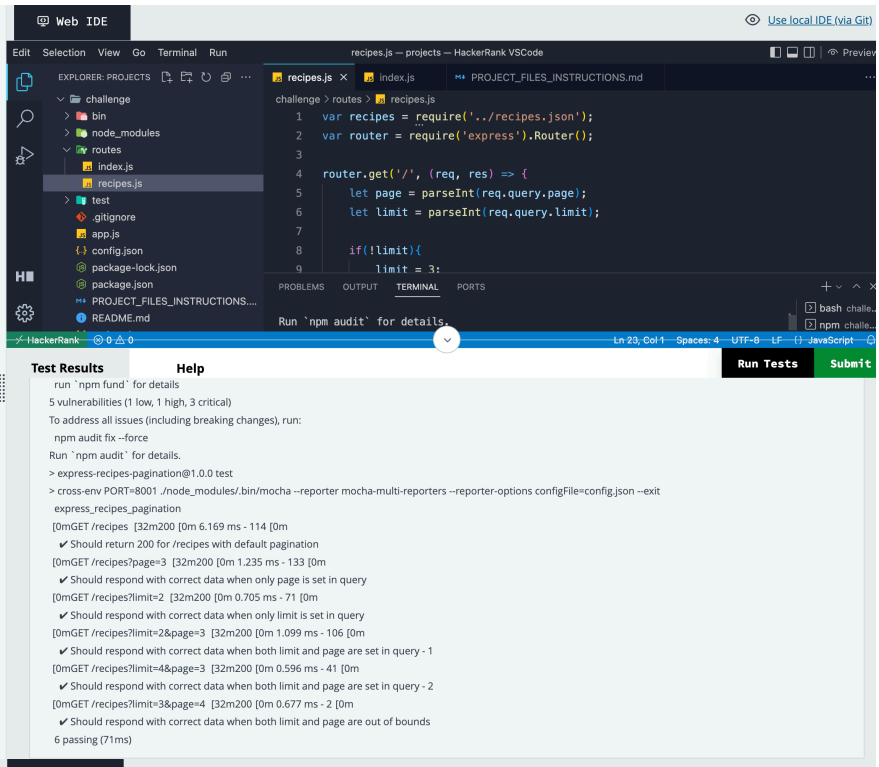
```
npm start
```

Test Copy

```
npm test
```

Install Copy

```
npm install
```




```
run `npm fund` for details
5 vulnerabilities (1 low, 1 high, 3 critical)
To address all issues (including breaking changes), run:
 npm audit fix --force
Run `npm audit` for details.
> express-recipes-pagination@1.0.0 test
> cross-env PORT=8001 ./node_modules/.bin/mocha --reporter mocha-multi-reporters --reporter-options configFile=config.json --exit
 express_recipes_pagination
[0mGET /recipes [32m200 [0m 6.169 ms - 114 [0m
  ✔ Should return 200 for /recipes with default pagination
[0mGET /recipes?page=3 [32m200 [0m 1.235 ms - 133 [0m
  ✔ Should respond with correct data when only page is set in query
[0mGET /recipes?limit=2 [32m200 [0m 0.705 ms - 71 [0m
  ✔ Should respond with correct data when only limit is set in query
[0mGET /recipes?limit=2&page=3 [32m200 [0m 1.099 ms - 106 [0m
  ✔ Should respond with correct data when both limit and page are set in query - 1
[0mGET /recipes?limit=4&page=3 [32m200 [0m 0.596 ms - 41 [0m
  ✔ Should respond with correct data when both limit and page are set in query - 2
[0mGET /recipes?limit=3&page=4 [32m200 [0m 0.677 ms - 2 [0m
  ✔ Should respond with correct data when both limit and page are out of bounds
 6 passing (71ms)
```


1. Node.js Express: Recipes Pagination

Your company is creating a new Recipe Management app. As the NodeJS developer in the company, you have been given the task to write a basic Express app that fetches the Recipe list from a data-store.

The request to the route /recipes returns all the paginated recipes **with default values of page and limit**. The query parameters that can be used to set the pagination criteria are:

- page: The page of the resource to be fetched. Defaults to 1. [NUMBER]
- limit: The number of items to be returned in a single page. Defaults to 3. [NUMBER]

▸ **Routes**

▸ **Examples**

Software Instructions

The question(s) requires **Node 18.15.0 LTS or above**.

- Download & Install Node.JS

Git Instructions

Use the following commands to work with this project

Run Copy

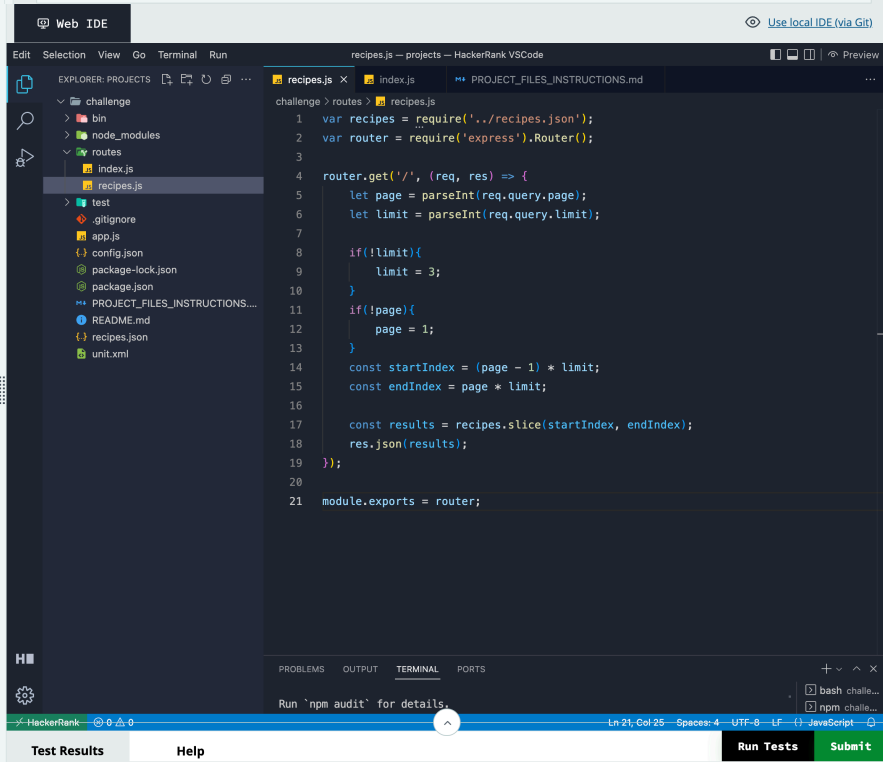
```
npm start
```

Test Copy

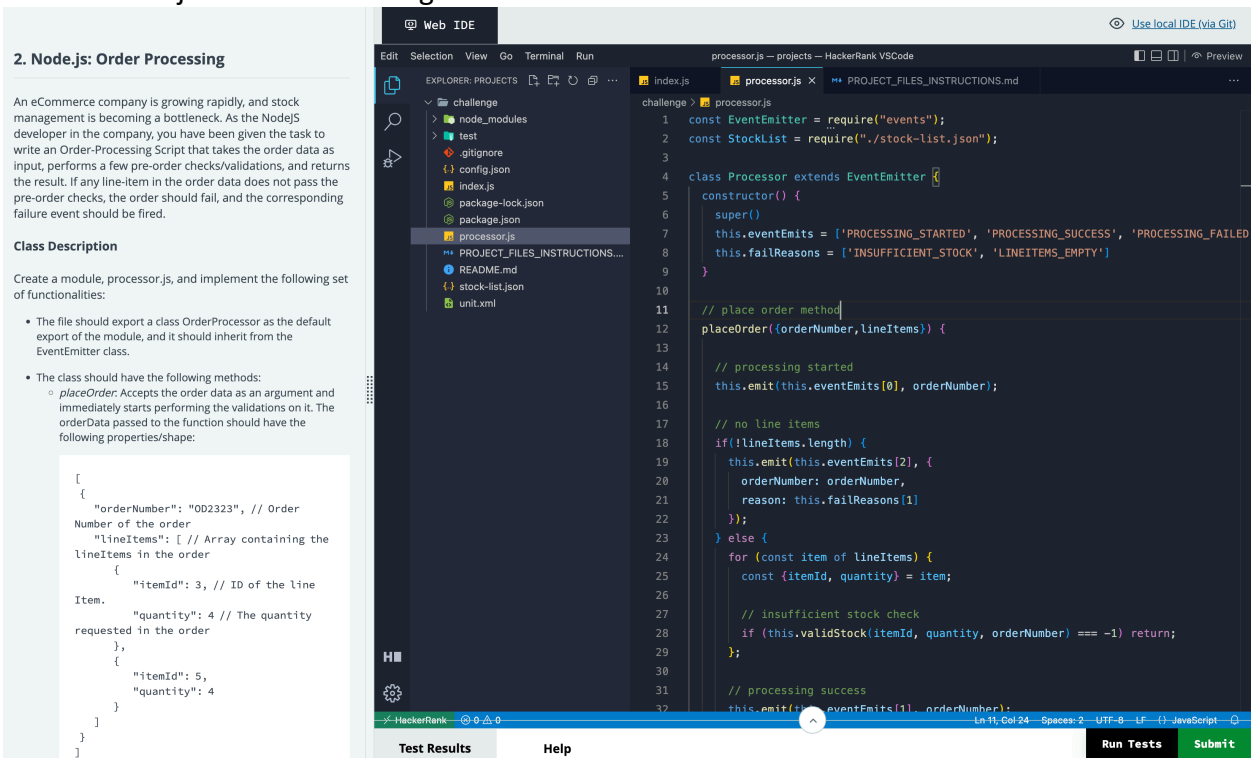
```
npm test
```

Install Copy

```
npm install
```

2. Node.js: Order Processing



2. Node.js: Order Processing

An eCommerce company is growing rapidly, and stock management is becoming a bottleneck. As the NodeJS developer in the company, you have been given the task to write an Order-Processing Script that takes the order data as input, performs a few pre-order checks/validations, and returns the result. If any line-item in the order data does not pass the pre-order checks, the order should fail, and the corresponding failure event should be fired.

Class Description

Create a module, processor.js, and implement the following set of functionalities:

- The file should export a class OrderProcessor as the default export of the module, and it should inherit from the EventEmitter class.
- The class should have the following methods:
  - *placeOrder*: Accepts the order data as an argument and immediately starts performing the validations on it. The orderData passed to the function should have the following properties/shape:

```
[
 {
   "orderNumber": "OD2323", // Order Number of the order
   "lineItems": [ // Array containing the lineItems in the order
      {
         "itemId": 3, // ID of the line Item.
         "quantity": 4 // The quantity requested in the order
      },
      {
         "itemId": 5,
         "quantity": 4
      }
   ]
 }
]
```

Web IDE

Use local IDE (via Git)

Edit Selection View Go Terminal Run — processor.js — projects — HackerRank VSCode — Preview

EXPLORER: PROJECTS

challenge
node_modules
test
.gitignore
config.json
index.js
package-lock.json
package.json
processor.js
PROJECT_FILES_INSTRUCTIONS....
README.md
stock-list.json
unit.xml

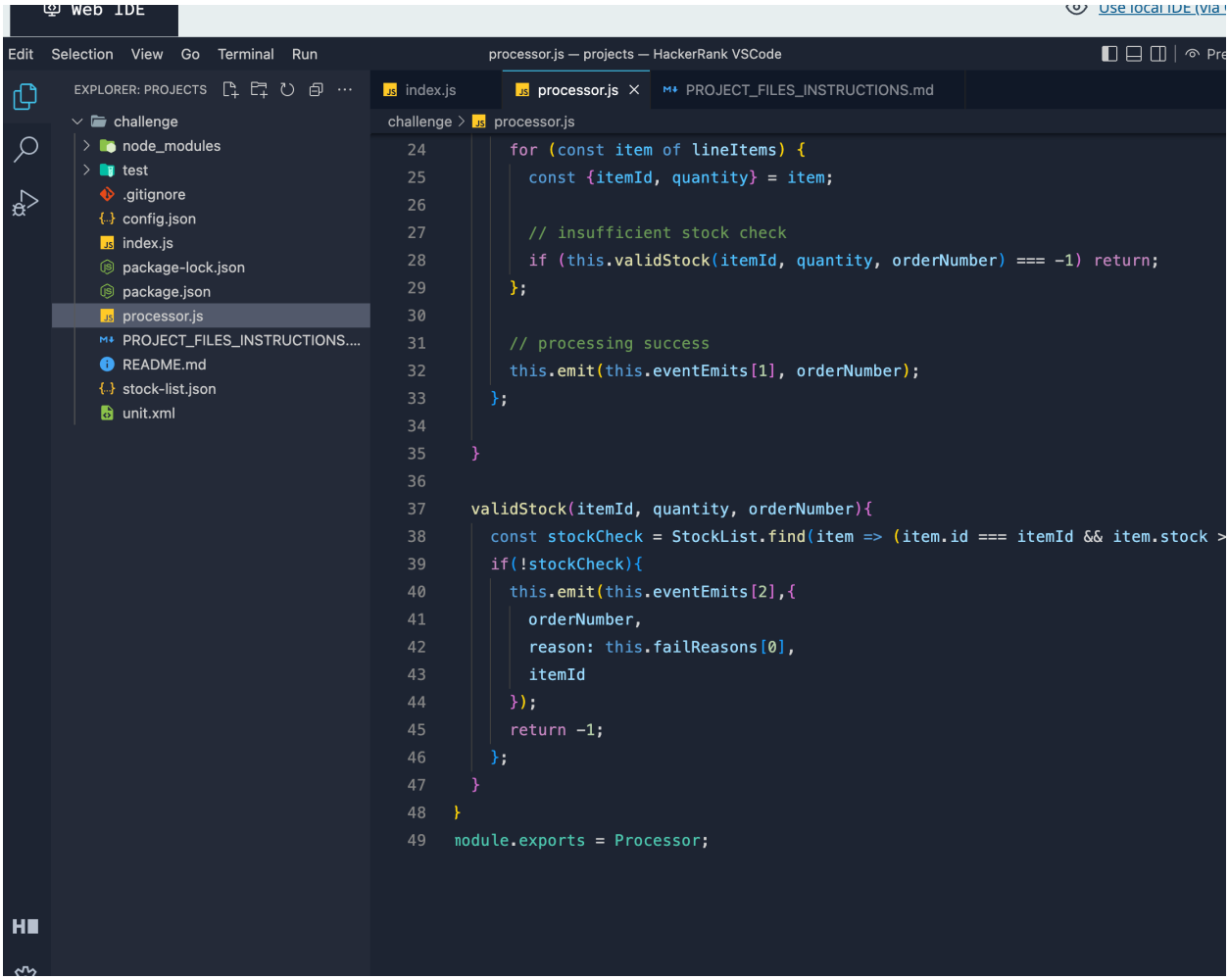
index.js | processor.js | PROJECT_FILES_INSTRUCTIONS.md

challenge > processor.js

```
1  const EventEmitter = require("events");
2  const StockList = require("./stock-list.json");
3
4  class Processor extends EventEmitter {
5    constructor() {
6      super()
7      this.eventEmits = ['PROCESSING_STARTED', 'PROCESSING_SUCCESS', 'PROCESSING_FAILED
8      this.failReasons = ['INSUFFICIENT_STOCK', 'LINEITEMS_EMPTY']
9    }
10
11   // place order method
12   placeOrder({orderNumber,lineItems}) {
13
14     // processing started
15     this.emit(this.eventEmits[0], orderNumber);
16
17     // no line items
18     if(!lineItems.length) {
19       this.emit(this.eventEmits[2], {
20         orderNumber: orderNumber,
21         reason: this.failReasons[1]
22       });
23     } else {
24       for (const item of lineItems) {
25         const {itemId, quantity} = item;
26
27         // insufficient stock check
28         if (this.validStock(itemId, quantity, orderNumber) === -1) return;
29       };
30
31       // processing success
32       this.emit(this.eventEmits[1], orderNumber);
```

HackerRank ⊗ 0 ⚠ 0 — Ln 11, Col 24 Spaces: 2 UTF-8 LF {} JavaScript

Test Results | Help | Run Tests | Submit


Web IDE
Edit Selection View Go Terminal Run
processor.js — projects — HackerRank VSCode
EXPLORER: PROJECTS
challenge
node_modules
test
.gitignore
config.json
index.js
package-lock.json
package.json
processor.js
PROJECT_FILES_INSTRUCTIONS....
README.md
stock-list.json
unit.xml
index.js
processor.js
PROJECT_FILES_INSTRUCTIONS.md
challenge > processor.js
for (const item of lineItems) {
const {itemId, quantity} = item;
// insufficient stock check
if (this.validStock(itemId, quantity, orderNumber) === -1) return;
};
// processing success
this.emit(this.eventEmits[1], orderNumber);
};
}
validStock(itemId, quantity, orderNumber){
const stockCheck = StockList.find(item => (item.id === itemId && item.stock >
if(!stockCheck){
this.emit(this.eventEmits[2],{
orderNumber,
reason: this.failReasons[0],
itemId
});
return -1;
};
}
}
module.exports = Processor;

Class Description

Create a module, processor.js, and implement the following set of functionalities:

- The file should export a class OrderProcessor as the default export of the module, and it should inherit from the EventEmitter class.
- The class should have the following methods:
  - *placeOrder*: Accepts the order data as an argument and immediately starts performing the validations on it. The orderData passed to the function should have the following properties/shape:

```
[
 {
    "orderNumber": "OD2323", // Order Number of the order
    "lineItems": [ // Array containing the lineItems in the order
        {
            "itemId": 3, // ID of the line Item.
            "quantity": 4 // The quantity requested in the order
        },
        {
            "itemId": 5,
            "quantity": 4
        }
    ]
 }
]
```

- The class should emit the following events:
  - PROCESSING_STARTED: Should be fired just before the validations are started for an order. The order number of the current order should be passed to the callback of the event handler.
  - PROCESSING_FAILED: Should be fired if, for any reason, the pre-order checks do not pass. The callback of the event handler will accept an object containing the following properties:
    - orderNumber: The orderNumber of the order that failed to process.
    - itemId: The ID of the first matching item that failed to process.
    - reason: The reason why the processing failed. It can be one of INSUFFICIENT_STOCK or LINEITEMS_EMPTY.
      - If the *lineItems* property is a blank array, the check should fail with the reason LINEITEMS_EMPTY. The *itemId* property need not be passed to the errorObject in this scenario.
      - If any of the *lineItems*' requested quantity is more than the matching stock property in the stockData array, the check should fail with the reason INSUFFICIENT_STOCK.
  - PROCESSING_SUCCESS: Should be fired if all the pre-order checks pass. The order number of the current order should be passed to the callback of the event handler.
- The file containing the current stock information, stock-list.json, is added to the root of the project and can be used for validation. An explanation of an entry in the JSON array is as follows:
  id: The ID of the item for which stock information is stored.
  stock: The current stock count of the item.

```
[
  {
    "id": 0,
    "stock": 4
  },
  {
    "id": 1,
    "stock": 12
  },
```

Web IDE

Use local IDE (via Git)

processor.js — projects — HackerRank VSCode

```
24      for (const item of lineItems) {
25        const {itemId, quantity} = item;
26
27        // insufficient stock check
28        if (this.validStock(itemId, quantity, orderNumber) === -1) return;
29      };
30
31      // processing success
32      this.emit(this.eventEmits[1], orderNumber);
33    };
34
35  }
36
37  validStock(itemId, quantity, orderNumber){
38    const stockCheck = StockList.find(item => (item.id === itemId && item.stock >= q
39    if(!stockCheck){
40      this.emit(this.eventEmits[2],{
41        orderNumber,
42        reason: this.failReasons[0],
43        itemId
44      });
45      return -1;
46    };
47  }
48  }
49  module.exports = Processor;
```

Test Results | Help | Run Tests | Submit

scenario.

- If any of the *lineItems*' requested quantity is more than the matching stock property in the stockData array, the check should fail with the reason INSUFFICIENT_STOCK.

- PROCESSING_SUCCESS: Should be fired if all the pre-order checks pass. The order number of the current order should be passed to the callback of the event handler.

- The file containing the current stock information, stock-list.json, is added to the root of the project and can be used for validation. An explanation of an entry in the JSON array is as follows:
  id: The ID of the item for which stock information is stored.
  stock: The current stock count of the item.

```
[
  {
    "id": 0,
    "stock": 4
  },
  {
    "id": 1,
    "stock": 12
  },
...
]
```

Note: It can be assumed that *lineitems* passed to the placeOrder function will always be an array, and the ID of the *lineItem* passed will always be present in the stock list. No extra code is required for these conditions.

▶ Example Implementation

Software Instructions

The question(s) requires **Node 18.15.0 LTS or above**.

- Download & Install Node.JS

Git Instructions

Web IDE

Use local IDE (via Git)

Edit Selection View Go Terminal Run — processor.js — projects — HackerRank VSCode — Preview

EXPLORER: PROJECTS

challenge
- node_modules
- test
- .gitignore
- config.json
- index.js
- package-lock.json
- package.json
- processor.js
- PROJECT_FILES_INSTRUCTIONS....
- README.md
- stock-list.json
- unit.xml

index.js | processor.js | PROJECT_FILES_INSTRUCTIONS.md

challenge > processor.js

```
24      for (const item of lineItems) {
25        const {itemId, quantity} = item;
26
27        // insufficient stock check
28        if (this.validStock(itemId, quantity, orderNumber) === -1) return;
29      };
30
31      // processing success
32      this.emit(this.eventEmits[1], orderNumber);
33    };
34
35  }
36
37  validStock(itemId, quantity, orderNumber){
38    const stockCheck = StockList.find(item => (item.id === itemId && item.stock >= q
39    if(!stockCheck){
40      this.emit(this.eventEmits[2],{
41        orderNumber,
42        reason: this.failReasons[0],
43        itemId
44      });
45      return -1;
46    };
47  }
48  }
49  module.exports = Processor;
```

HackerRank 0 0 — Ln 11, Col 24 Spaces: 2 UTF-8 LF {} JavaScript

Test Results | Help | Run Tests | Submit

Note: It can be assumed that *lineitems* passed to the placeOrder function will always be an array, and the ID of the *lineItem* passed will always be present in the stock list. No extra code is required for these conditions.

▼ Example Implementation

```
const OrderProcessor = require('./order-
processor');

const orderProcessor = new OrderProcessor();

orderProcessor.on('PROCESSING_STARTED',
(orderNumber) => {
    console.log(`Pre Order Checks Running for
${orderNumber}`)
});

orderProcessor.on('PROCESSING_FAILED',
(failureData) => {
    console.log(`Failed Order`);
    console.log(`- OrderNumber:
${failureData.orderNumber}`);
    console.log(`- Reason:
${failureData.reason}`);
    console.log(`- ItemId:
${failureData.itemId}`);
})

orderProcessor.on('PROCESSING_SUCCESS',
(orderNumber) => {
    console.log(`Pre Order Checks Passed for
${orderNumber}`)
})

orderProcessor.placeOrder({
    orderNumber: 'OD2323',
    lineItems: [
        {
            itemId: 3,
            quantity: 4
        },
        {
            itemId: 5,
            quantity: 4
        }
    ]
});
```

Web IDE

Use local IDE (via Git)

Edit Selection View Go Terminal Run — processor.js — projects — HackerRank VSCode — Preview

EXPLORER: PROJECTS

challenge
- node_modules
- test
- .gitignore
- config.json
- index.js
- package-lock.json
- package.json
- processor.js
- PROJECT_FILES_INSTRUCTIONS....
- README.md
- stock-list.json
- unit.xml

index.js | processor.js | PROJECT_FILES_INSTRUCTIONS.md

challenge > processor.js

```
2   const StockList = require("./stock-list.json");
3
4   class Processor extends EventEmitter {
5     constructor() {
6       super()
7       this.eventEmits = ['PROCESSING_STARTED', 'PROCESSING_SUCCESS', 'PROCESSING_FAILED
8       this.failReasons = ['INSUFFICIENT_STOCK', 'LINEITEMS_EMPTY']
9     }
10
11    // place order method
12    placeOrder({orderNumber,lineItems}) {
13
14      // processing started
15      this.emit(this.eventEmits[0], orderNumber);
16
17      // no line items
18      if(!lineItems.length) {
19        this.emit(this.eventEmits[2], {
20          orderNumber: orderNumber,
21          reason: this.failReasons[1]
22        });
23      } else {
24        for (const item of lineItems) {
25          const {itemId, quantity} = item;
26
27          // insufficient stock check
28          if (this.validStock(itemId, quantity, orderNumber) === -1) return;
29        };
30
31        // processing success
32        this.emit(this.eventEmits[1], orderNumber);
```

HackerRank 0 0 — Ln 11, Col 24 Spaces: 2 UTF-8 LF {} JavaScript

Test Results | Help | Run Tests | Submit

2. Node.js: Order Processing

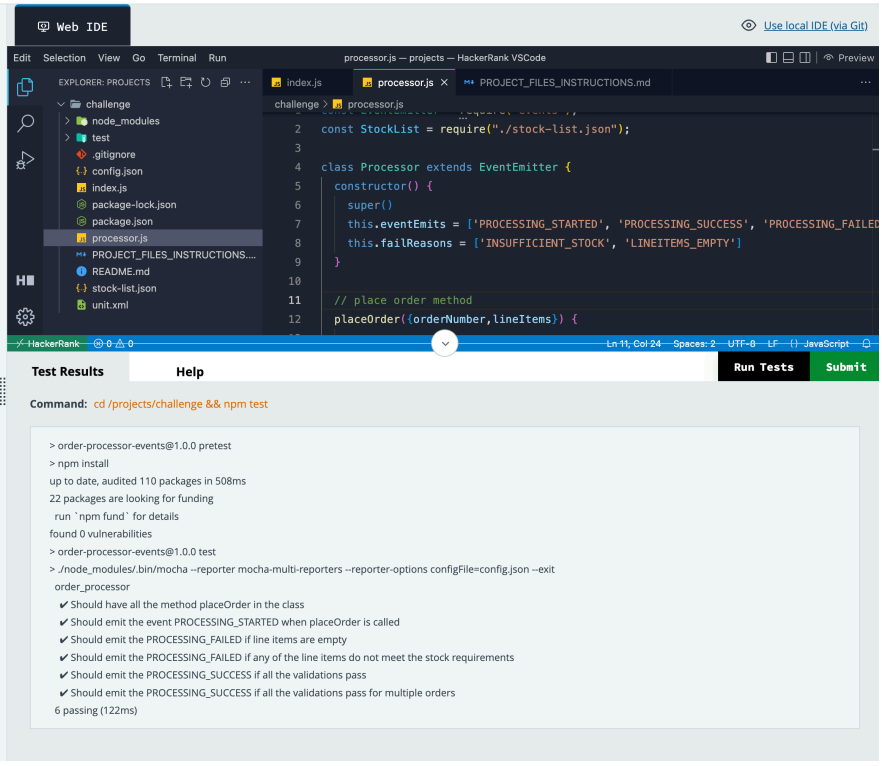
An eCommerce company is growing rapidly, and stock management is becoming a bottleneck. As the NodeJS developer in the company, you have been given the task to write an Order-Processing Script that takes the order data as input, performs a few pre-order checks/validations, and returns the result. If any line-item in the order data does not pass the pre-order checks, the order should fail, and the corresponding failure event should be fired.

Class Description

Create a module, processor.js, and implement the following set of functionalities:

- The file should export a class OrderProcessor as the default export of the module, and it should inherit from the EventEmitter class.
- The class should have the following methods:
  - *placeOrder*: Accepts the order data as an argument and immediately starts performing the validations on it. The orderData passed to the function should have the following properties/shape:

```
[
 {
    "orderNumber": "OD2323", // Order Number of the order
    "lineItems": [ // Array containing the lineItems in the order
        {
            "itemId": 3, // ID of the line Item.
            "quantity": 4 // The quantity requested in the order
        },
        {
            "itemId": 5,
            "quantity": 4
        }
    ]
 }
]
```



Command: cd /projects/challenge && npm test

```
> order-processor-events@1.0.0 pretest
> npm install
up to date, audited 110 packages in 508ms
22 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
> order-processor-events@1.0.0 test
> ./node_modules/.bin/mocha --reporter mocha-multi-reporters --reporter-options configFile=config.json --exit
  order_processor
    ✔ Should have all the method placeOrder in the class
    ✔ Should emit the event PROCESSING_STARTED when placeOrder is called
    ✔ Should emit the PROCESSING_FAILED if line items are empty
    ✔ Should emit the PROCESSING_FAILED if any of the line items do not meet the stock requirements
    ✔ Should emit the PROCESSING_SUCCESS if all the validations pass
    ✔ Should emit the PROCESSING_SUCCESS if all the validations pass for multiple orders
  6 passing (122ms)
```


```
    console.log(`- ItemId: ${failureData.itemId}`);
})

orderProcessor.on('PROCESSING_SUCCESS', (orderNumber) => {
    console.log(`Pre Order Checks Passed for ${orderNumber}`)
})

orderProcessor.placeOrder({
    orderNumber: 'OD2323',
    lineItems: [
        {
            itemId: 3,
            quantity: 4
        },
        {
            itemId: 5,
            quantity: 4
        }
    ]
});
```


Software Instructions

The question(s) requires **Node 18.15.0 LTS or above**.

- Download & Install Node.JS

Git Instructions

Use the following commands to work with this project

Run Copy

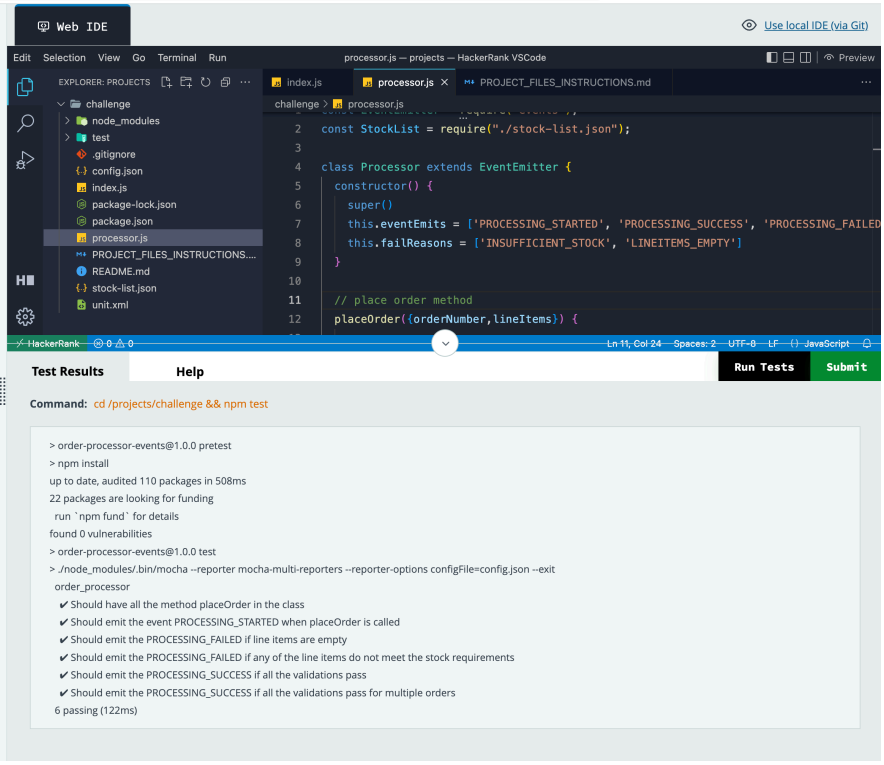
```
npm start
```

Test Copy

```
npm test
```

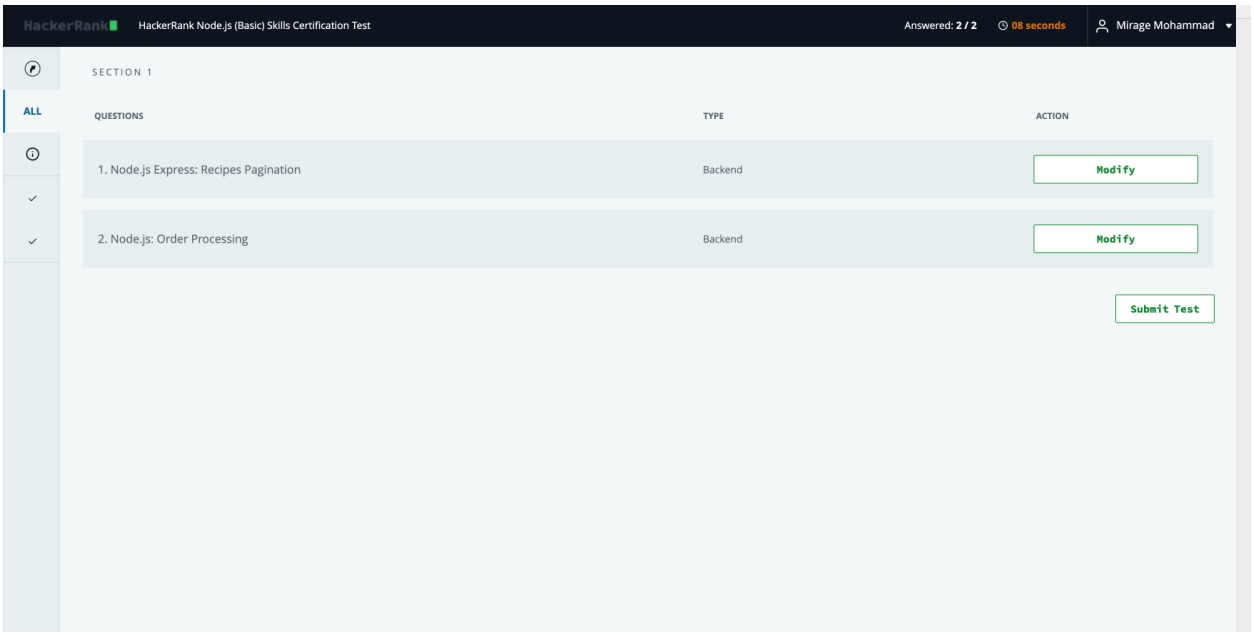
Install Copy

```
npm install
```

Command: cd /projects/challenge && npm test

```
> order-processor-events@1.0.0 pretest
> npm install
up to date, audited 110 packages in 508ms
22 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
> order-processor-events@1.0.0 test
> ./node_modules/.bin/mocha --reporter mocha-multi-reporters --reporter-options configFile=config.json --exit
  order_processor
    ✔ Should have all the method placeOrder in the class
    ✔ Should emit the event PROCESSING_STARTED when placeOrder is called
    ✔ Should emit the PROCESSING_FAILED if line items are empty
    ✔ Should emit the PROCESSING_FAILED if any of the line items do not meet the stock requirements
    ✔ Should emit the PROCESSING_SUCCESS if all the validations pass
    ✔ Should emit the PROCESSING_SUCCESS if all the validations pass for multiple orders
  6 passing (122ms)
```


HackerRank
HackerRank Node.js (Basic) Skills Certification Test
Answered: 2 / 2
08 seconds
Mirage Mohammad
SECTION 1
ALL
QUESTIONS
TYPE
ACTION
1. Node.js Express: Recipes Pagination
Backend
Modify
2. Node.js: Order Processing
Backend
Modify
Submit Test