



- Write Cypress API tests for the following API (see below delimited by md)

- Use object destructuring for the request's response properties so that we get only the ones used in the assertions
- Do not use the word should in the tests' descriptions
- When using cy.request, use it in the following form:  
cy.request('verb', 'api-url?query-string=value')
- Get the API URL using the Cypress.env method
- When defining a name for the complete API URL variable, call it CUSTOMERS\_API\_URL
- Do not use semi-colons at the end of any JavaScript expression
- When defining the complete API Url, use template literals.
- For the tests that make sense, do not assert on the data returned. Instead, assert on its structure. For the ones that make sense, assert on the returned data as well (e.g., filtering)
- Test one behavior at a time. Do not mix the same behavior in the same test.

md  
# EngageSphere Server

This is a sample backend API that offers one endpoint

## GET /customers

This endpoint accepts four query parameters, as listed

### Query parameters

- `page` (optional; current page - default is 1)
- `limit` (optional; number of customers to be shown)
- `size` (optional; filter customers by size: 'Small')
- `industry` (optional; filter customers by business)

Below is an example of how the request URL would look

`http://localhost:3001/customers?page=2&limit=10&size=50`

### Response body

If there are customers in the database, the following JSON

```
{  
  "customers": [  
    {  
      "id": 1,  
      "name": "Thompson, Zboncak and Mueller",  
      "employees": 850,  
      "contactInfo": null,  
      "size": "Medium",  
      "industry": "Technology",  
      "address": {  
        "street": "988 Kimberly Fort Apt. 921",  
        "city": "Lake Tracy",  
        "state": "Connecticut",  
        "zipCode": "07115",  
        "country": "United States of America"  
      }  
    }  
  ]  
}
```

```

    }
},
{
  "id": 2,
  "name": "Americas Inc.",
  "employees": 200,
  "contactInfo": {
    "name": "John Smith",
    "email": "jsmith@americasinc.com"
  },
  "size": "Medium",
  "industry": "Technology",
  "address": {
    "street": "5099 Murray Inlet",
    "city": "South Tiffany",
    "state": "Kentucky",
    "zipCode": "08496",
    "country": "United States of America"
  }
},
...
],
"pageInfo": {
  "currentPage": 2,
  "totalPages": 20,
  "totalCustomers": 199
}
}
}

**Notes:**

```

- The **contactInfo** and **address** values are `null`
- The `size` attribute in the response for each customer:
  - **Small** when the **Number of employees** is less than or equal to 100
  - **Medium** when it is greater or equal to 100 and less than or equal to 1000
  - **Enterprise** when it is greater than or equal to 1000 and less than or equal to 5000
  - **Large Enterprise** when it is greater than or equal to 5000 and less than or equal to 10000
  - **Very Large Enterprise** otherwise;
- The `industry` attribute corresponds to the business industry
- The API responds with a `400 Bad Request` status code if the input is invalid

#### ## Swagger

The API also has a Swagger documentation, which can be found at [https://api.engage Sphere.com/swagger](#)

< 7/7 >



Here are Cypress API tests for the EngageSphere API, each focused on a single behavior and following the guidelines you provided:

```

javascript
Copiar código

describe('EngageSphere API - GET /customers', () => {
  const CUSTOMERS_API_URL = `${Cypress.env('API_URL')}/customers`

  it('verifies structure of default customer list response', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}`).then((response) => {
      const { status, body: { customers, pageInfo } } = response

      expect(status).to.eq(200)
      expect(customers).to.be.an('array')
      expect(pageInfo).to.be.an('object')

      customers.forEach((customer) => {
        expect(customer.id).to.be.a('number')
        expect(customer.name).to.be.a('string')
        expect(customer.employees).to.be.a('number')
        expect(customer.contactInfo.name).to.be.a('string')
        expect(customer.contactInfo.email).to.be.a('string')
        expect(customer.size).to.be.oneOf(['Small', 'Medium', 'Enterprise', 'Large Enterprise', 'Very Large Enterprise'])
        expect(customer.industry).to.be.a('string')
        expect(customer.address.street).to.be.a('string')
        expect(customer.address.city).to.be.a('string')
        expect(customer.address.state).to.be.a('string')
        expect(customer.address.zipCode).to.be.a('string')
        expect(customer.address.country).to.be.a('string')
      })
    })
  })
})

```

```
    expect(customer).to.have.all.keys('id', 'name', 'employees', 'contactInfo',
  })
}

expect(pageInfo).to.have.all.keys('currentPage', 'totalPages', 'totalCustomers',
  })
}

it('fetches customers with specific page and validates current page in pageInfo',
  cy.request('GET', `${CUSTOMERS_API_URL}?page=2`).then((response) => {
  const { body: { pageInfo: { currentPage } } } = response
  expect(currentPage).to.eq(2)
})
}

it('limits customer records per page and verifies response length', () => {
  cy.request('GET', `${CUSTOMERS_API_URL}?limit=5`).then((response) => {
  const { body: { customers } } = response
  expect(customers).to.have.length(5)
})
})

it('fetches customers filtered by size and validates size of each customer', () => {
  cy.request('GET', `${CUSTOMERS_API_URL}?size=Medium`).then((response) => {
  const { body: { customers } } = response
  customers.forEach(({ size }) => {
    expect(size).to.eq('Medium')
  })
})
})

it('fetches customers filtered by industry and validates industry of each customer',
  cy.request('GET', `${CUSTOMERS_API_URL}?industry=Technology`).then((response) => {
  const { body: { customers } } = response
  customers.forEach(({ industry }) => {
    expect(industry).to.eq('Technology')
  })
})
})

it('validates error response structure for invalid page parameter', () => {
  cy.request({
    method: 'GET',
    url: `${CUSTOMERS_API_URL}?page=-1`,
    failOnStatusCode: false
  }).then((response) => {
  const { status, body: { error } } = response
  expect(status).to.eq(400)
  expect(error).to.be.a('string')
})
})

it('validates error response structure for unsupported size parameter', () => {
  cy.request({
    method: 'GET',
    url: `${CUSTOMERS_API_URL}?size=InvalidSize`,
    failOnStatusCode: false
  }).then((response) => {
  const { status, body: { error } } = response
  expect(status).to.eq(400)
  expect(error).to.be.a('string')
})
})
```

```

it('fetches customers on specific page with limit and verifies structure and length', () => {
  cy.request('GET', `${CUSTOMERS_API_URL}?page=3&limit=3`).then((response) => {
    const { status, body: { customers, pageInfo: { currentPage } } } = response

    expect(status).to.eq(200)
    expect(customers).to.be.an('array').with.lengthOf(3)
    expect(currentPage).to.eq(3)

    customers.forEach((customer) => {
      expect(customer).to.have.all.keys('id', 'name', 'employees', 'contactInfo')
    })
  })
})
}

```

## Explanation of Each Test Case

1. **Verifies Structure of Default Customer List Response:**
  - Requests the default customer list and checks the structure of each customer object and the `pageInfo` object.
2. **Fetches Customers with Specific Page and Validates Current Page in PageInfo:**
  - Requests customers on page 2 and validates that `currentPage` in `pageInfo` is set to 2.
3. **Limits Customer Records per Page and Verifies Response Length:**
  - Requests a limited number of customers (`limit=5`) and confirms that the response contains exactly 5 customers.
4. **Fetches Customers Filtered by Size and Validates Size of Each Customer:**
  - Filters by `size=Medium` and verifies that each returned customer has `size` set to "Medium."
5. **Fetches Customers Filtered by Industry and Validates Industry of Each Customer:**
  - Filters by `industry=Technology` and confirms that each returned customer has `industry` set to "Technology."
6. **Validates Error Response Structure for Invalid Page Parameter:**
  - Sends an invalid `page` parameter and checks for a 400 error with the `error` message in the response.
7. **Validates Error Response Structure for Unsupported Size Parameter:**
  - Uses an unsupported `size` parameter and expects a 400 error with the `error` message in the response.
8. **Fetches Customers on Specific Page with Limit and Verifies Structure and Length:**
  - Combines `page=3` and `limit=3`, verifies the structure of each customer and confirms `currentPage` is 3, and checks that exactly 3 customers are returned.

Each test case is written to isolate and validate a single behavior, either by checking the structure or specific data returned when filters are applied. This approach ensures focused testing and adherence to the specifications.

Update the negative scenarios to check on the returned error message.

Use the previous documentation to understand exactly what the error messages should be for different scenarios.



Based on the documentation provided, I have updated the negative test scenarios to check the returned error message, ensuring they match the specified response. Here is the updated test suite:

```
javascript Copiar código

describe('EngageSphere API - GET /customers', () => {
  const CUSTOMERS_API_URL = `${Cypress.env('API_URL')}/customers`

  it('verifies structure of default customer list response', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}`).then((response) => {
      const { status, body: { customers, pageInfo } } = response

      expect(status).to.eq(200)
      expect(customers).to.be.an('array')
      expect(pageInfo).to.be.an('object')

      customers.forEach((customer) => {
        expect(customer).to.have.all.keys('id', 'name', 'employees', 'contactInfo')
      })

      expect(pageInfo).to.have.all.keys('currentPage', 'totalPages', 'totalCustomers')
    })
  })

  it('fetches customers with specific page and validates current page in pageInfo', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?page=2`).then((response) => {
      const { body: { pageInfo: { currentPage } } } = response
      expect(currentPage).to.eq(2)
    })
  })

  it('limits customer records per page and verifies response length', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?limit=5`).then((response) => {
      const { body: { customers } } = response
      expect(customers).to.have.length(5)
    })
  })

  it('fetches customers filtered by size and validates size of each customer', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?size=Medium`).then((response) => {
      const { body: { customers } } = response
      customers.forEach(({ size }) => {
        expect(size).to.eq('Medium')
      })
    })
  })

  it('fetches customers filtered by industry and validates industry of each customer', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?industry=Technology`).then((response) => {
      const { body: { customers } } = response
      customers.forEach(({ industry }) => {
        expect(industry).to.eq('Technology')
      })
    })
  })

  it('validates error response structure and message for invalid page parameter', () => {
    cy.request({
      method: 'GET',
      url: `${CUSTOMERS_API_URL}?page=-1`,
      failOnStatusCode: false
    })
  })
})
```

```

    }).then((response) => {
      const { status, body: { error } } = response
      expect(status).to.eq(400)
      expect(error).to.eq('Invalid request parameters')
    })
  })

it('validates error response structure and message for unsupported size parameter', () => {
  cy.request({
    method: 'GET',
    url: `${CUSTOMERS_API_URL}?size=invalidSize`,
    failOnStatusCode: false
  }).then((response) => {
    const { status, body: { error } } = response
    expect(status).to.eq(400)
    expect(error).to.eq('Invalid request parameters')
  })
})

it('validates error response structure and message for invalid industry parameter', () => {
  cy.request({
    method: 'GET',
    url: `${CUSTOMERS_API_URL}?industry=UnknownIndustry`,
    failOnStatusCode: false
  }).then((response) => {
    const { status, body: { error } } = response
    expect(status).to.eq(400)
    expect(error).to.eq('Invalid request parameters')
  })
})

it('fetches customers on specific page with limit and verifies structure and length', () => {
  cy.request('GET', `${CUSTOMERS_API_URL}?page=3&limit=3`).then((response) => {
    const { status, body: { customers, pageInfo: { currentPage } } } = response

    expect(status).to.eq(200)
    expect(customers).to.be.an('array').with.lengthOf(3)
    expect(currentPage).to.eq(3)

    customers.forEach((customer) => {
      expect(customer).to.have.all.keys('id', 'name', 'employees', 'contactInfo')
    })
  })
})
}

```

## Explanation of Negative Test Scenarios

### 1. Invalid Page Parameter:

- Sends an invalid `page` parameter (`page=-1`) and expects a 400 status code.
- Checks that the `error` message is "Invalid request parameters" as specified in the documentation.

### 2. Unsupported Size Parameter:

- Uses an unsupported `size` parameter (`size=InvalidSize`) and expects a 400 status code.
- Asserts that the `error` message is "Invalid request parameters".

### 3. Invalid Industry Parameter:

- Uses an invalid `industry` parameter (`industry=UnknownIndustry`) to trigger a 400 error.

- Validates that the `error` message returned is "Invalid request parameters".

Each negative test is updated to verify that the `error` message matches the expected string "Invalid request parameters" based on the provided documentation. This ensures consistency with the API's expected error handling behavior.

Now, separate the error scenarios in their own context.



Here's the test suite with the error scenarios separated into their own context :

```
javascript Copiar código

describe('EngageSphere API - GET /customers', () => {
  const CUSTOMERS_API_URL = `${Cypress.env('API_URL')}/customers`

  context('Successful scenarios', () => {
    it('verifies structure of default customer list response', () => {
      cy.request('GET', `${CUSTOMERS_API_URL}`).then((response) => {
        const { status, body: { customers, pageInfo } } = response

        expect(status).to.eq(200)
        expect(customers).to.be.an('array')
        expect(pageInfo).to.be.an('object')

        customers.forEach((customer) => {
          expect(customer).to.have.all.keys('id', 'name', 'employees', 'contactInfo')
        })

        expect(pageInfo).to.have.all.keys('currentPage', 'totalPages', 'totalCustomers')
      })
    })
  })

  it('fetches customers with specific page and validates current page in pageInfo', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?page=2`).then((response) => {
      const { body: { pageInfo: { currentPage } } } = response
      expect(currentPage).to.eq(2)
    })
  })

  it('limits customer records per page and verifies response length', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?limit=5`).then((response) => {
      const { body: { customers } } = response
      expect(customers).to.have.length(5)
    })
  })

  it('fetches customers filtered by size and validates size of each customer', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?size=Medium`).then((response) => {
      const { body: { customers } } = response
      customers.forEach(({ size }) => {
        expect(size).to.eq('Medium')
      })
    })
  })

  it('fetches customers filtered by industry and validates industry of each customer', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?industry=Technology`).then((response) => {
      const { body: { customers } } = response
      customers.forEach(({ industry }) => {
        expect(industry).to.eq('Technology')
      })
    })
  })
})
```

```

        })
    })
})

it('fetches customers on specific page with limit and verifies structure and length', () => {
    cy.request('GET', `${CUSTOMERS_API_URL}?page=3&limit=3`).then((response) => {
        const { status, body: { customers, pageInfo: { currentPage } } } = response

        expect(status).to.eq(200)
        expect(customers).to.be.an('array').with.lengthOf(3)
        expect(currentPage).to.eq(3)

        customers.forEach((customer) => {
            expect(customer).to.have.all.keys('id', 'name', 'employees', 'contactInfo')
        })
    })
}

context('Error scenarios', () => {
    it('validates error response structure and message for invalid page parameter', () => {
        cy.request({
            method: 'GET',
            url: `${CUSTOMERS_API_URL}?page=-1`,
            failOnStatusCode: false
        }).then((response) => {
            const { status, body: { error } } = response
            expect(status).to.eq(400)
            expect(error).to.eq('Invalid request parameters')
        })
    })

    it('validates error response structure and message for unsupported size parameter', () => {
        cy.request({
            method: 'GET',
            url: `${CUSTOMERS_API_URL}?size=InvalidSize`,
            failOnStatusCode: false
        }).then((response) => {
            const { status, body: { error } } = response
            expect(status).to.eq(400)
            expect(error).to.eq('Invalid request parameters')
        })
    })

    it('validates error response structure and message for invalid industry parameter', () => {
        cy.request({
            method: 'GET',
            url: `${CUSTOMERS_API_URL}?industry=UnknownIndustry`,
            failOnStatusCode: false
        }).then((response) => {
            const { status, body: { error } } = response
            expect(status).to.eq(400)
            expect(error).to.eq('Invalid request parameters')
        })
    })
})
}

```

## Explanation

- The test suite is divided into two `context` sections: **Successful scenarios** and **Error scenarios**.

- **Successful scenarios** context contains tests that validate normal operation, such as verifying response structure and specific data when filtering.
- **Error scenarios** context contains tests for invalid input parameters, checking both the status code and specific error messages ("Invalid request parameters") as per the API documentation.

This organization makes the suite cleaner and separates positive and negative test cases, improving readability and maintainability.

Write a scenario for an invalid limit.

Use the following error message: Invalid page or limit. Both must be positive numbers.'

< 2/2 >



Here's the updated test suite with a scenario for an invalid `limit` parameter in the **Error scenarios** context, checking the specific error message:

```
javascript Copiar código

describe('EngageSphere API - GET /customers', () => {
  const CUSTOMERS_API_URL = `${Cypress.env('API_URL')}/customers`

  context('Successful scenarios', () => {
    it('verifies structure of default customer list response', () => {
      cy.request('GET', `${CUSTOMERS_API_URL}`).then((response) => {
        const { status, body: { customers, pageInfo } } = response

        expect(status).to.eq(200)
        expect(customers).to.be.an('array')
        expect(pageInfo).to.be.an('object')

        customers.forEach((customer) => {
          expect(customer).to.have.all.keys('id', 'name', 'employees', 'contactInfo')
        })

        expect(pageInfo).to.have.all.keys('currentPage', 'totalPages', 'totalCustomers')
      })
    })

    it('fetches customers with specific page and validates current page in pageInfo', () => {
      cy.request('GET', `${CUSTOMERS_API_URL}?page=2`).then((response) => {
        const { body: { pageInfo: { currentPage } } } = response
        expect(currentPage).to.eq(2)
      })
    })

    it('limits customer records per page and verifies response length', () => {
      cy.request('GET', `${CUSTOMERS_API_URL}?limit=5`).then((response) => {
        const { body: { customers } } = response
        expect(customers).to.have.length(5)
      })
    })

    it('fetches customers filtered by size and validates size of each customer', () => {
      cy.request('GET', `${CUSTOMERS_API_URL}?size=Medium`).then((response) => {
        const { body: { customers } } = response
        customers.forEach(({ size }) => {
          expect(size).to.eq('Medium')
        })
      })
    })
  })
})
```

```
        })
    })
})

it('fetches customers filtered by industry and validates industry of each customer')
cy.request('GET', `${CUSTOMERS_API_URL}?industry=Technology`).then((response) =>
  const { body: { customers } } = response
  customers.forEach(({ industry }) => {
    expect(industry).to.eq('Technology')
  })
)
})

it('fetches customers on specific page with limit and verifies structure and length')
cy.request('GET', `${CUSTOMERS_API_URL}?page=3&limit=3`).then((response) => {
  const { status, body: { customers, pageInfo: { currentPage } } } = response

  expect(status).to.eq(200)
  expect(customers).to.be.an('array').with.lengthOf(3)
  expect(currentPage).to.eq(3)

  customers.forEach((customer) => {
    expect(customer).to.have.all.keys('id', 'name', 'employees', 'contactInfo')
  })
)
})

context('Error scenarios', () => {
  it('validates error response structure and message for invalid page parameter')
  cy.request({
    method: 'GET',
    url: `${CUSTOMERS_API_URL}?page=-1`,
    failOnStatusCode: false
  }).then((response) => {
    const { status, body: { error } } = response
    expect(status).to.eq(400)
    expect(error).to.eq('Invalid request parameters')
  })
})

it('validates error response structure and message for unsupported size parameter')
cy.request({
  method: 'GET',
  url: `${CUSTOMERS_API_URL}?size=InvalidSize`,
  failOnStatusCode: false
}).then((response) => {
  const { status, body: { error } } = response
  expect(status).to.eq(400)
  expect(error).to.eq('Invalid request parameters')
})
})

it('validates error response structure and message for invalid industry parameter')
cy.request({
  method: 'GET',
  url: `${CUSTOMERS_API_URL}?industry=UnknownIndustry`,
  failOnStatusCode: false
}).then((response) => {
  const { status, body: { error } } = response
  expect(status).to.eq(400)
  expect(error).to.eq('Invalid request parameters')
})
```

```
        })

    it('validates error response structure and message for invalid limit parameter', () => {
      cy.request({
        method: 'GET',
        url: `${CUSTOMERS_API_URL}?limit=-5`,
        failOnStatusCode: false
      }).then((response) => {
        const { status, body: { error } } = response
        expect(status).to.eq(400)
        expect(error).to.eq('Invalid page or limit. Both must be positive numbers.')
      })
    })
  })
})
```

### Explanation of the New Test Case

- Invalid Limit Parameter:

- Sends an invalid `limit` parameter (`limit=-5`) and expects a 400 status code.
- Checks that the `error` message is "Invalid page or limit. Both must be positive numbers." as specified.

This test ensures that the API handles invalid `limit` values correctly and returns the appropriate error message, keeping it consistent with the provided documentation.

Write two new test cases (just the new ones, no need to write the whole suite), for cases where page and limit are equal to zero (i.e., 0).

Here are the two new test cases for scenario where `page` and `limit` are set to zero (0):

```
javascipt  Copiar código

it('validates error response structure and message for page parameter equal to zero', () => {
  cy.request({
    method: 'GET',
    url: `${CUSTOMERS_API_URL}?page=0`,
    failOnStatusCode: false
  }).then((response) => {
    const { status, body: { error } } = response
    expect(status).to.eq(400)
    expect(error).to.eq('Invalid page or limit. Both must be positive numbers.')
  })
})

it('validates error response structure and message for limit parameter equal to zero', () => {
  cy.request({
    method: 'GET',
    url: `${CUSTOMERS_API_URL}?limit=0`,
    failOnStatusCode: false
  }).then((response) => {
    const { status, body: { error } } = response
    expect(status).to.eq(400)
    expect(error).to.eq('Invalid page or limit. Both must be positive numbers.')
  })
})
```

#### EXPLANATION OF EACH NEW TEST CASE

##### 1. Page Parameter Equal to Zero:

- Sends a request with `page=0` to test the response when `page` is zero.
- Expects a 400 status code and checks that the error message is "Invalid page or limit. Both must be positive numbers."

##### 2. Limit Parameter Equal to Zero:

- Sends a request with `limit=0` to test the response when `limit` is zero.
- Expects a 400 status code and checks that the error message is "Invalid page or limit. Both must be positive numbers."

These tests ensure that the API handles cases where `page` or `limit` is zero and returns the correct error message.

⟳ ⏷ ⏸ ⏹ ⏵

