통합 테스트(Integration Test)에 대해서

통합 테스트란 ...?

통합테스트는 모듈을 통합하는 단계에서 수행하는 테스트입니다.
단위 테스트를 먼저 수행하여 모듈들이 잘 작동되는 것을 확인했다면
이제 이 모듈들을 연동하여서 테스트를 수행합니다.
이렇게 연동해서 테스트하는 것이 통합테스트입니다.

통합 테스트를 하는 이유...?

1. 모듈들의 상호 작용이 잘 이루어지는지 검증하기 위해서
2. 통합하는 과정에서 발생할 수 있는 오류를 찾기 위해서

Supertest 란 무엇인가요 ?

nodejs http 서버를 테스트하기 위해 만들어진 모듈입니다.
supertest 모듈을 이용해서 통합 테스트를 쉽게 구현할 수 있습니다.

Supertest 를 이용해서 통합 테스트 구현하는 법

nodejs http 서버를 테스트하기 위해 만들어진 모듈입니다.
supertest 모듈을 이용해서 통합 테스트를 쉽게 구현할 수 있습니다.

```
const request = require('supertest');
const express = require('express');

const app = express();

app.get('/user', function(req, res) {
  res.status(200).json({ name: 'john' });
});
```

→ 클라이언트에서 들어오는 요청을 처리하는 원본 소스

```
request(app)
  .get('/user')
  .expect('Content-Type', /json/)
  .expect('Content-Length', '15')
  .expect(200)
  .end(function(err, res) {
    if (err) throw err;
  });
```

위에 원본 소스를 위한 통합 테스트 소스

Create을 위한 통합 테스트

해야 할 일은 ???

Product 데이터를
데이터베이스에 넣기

통합 테스트 작성

```
it("POST /api/products", async () => {
  const response = await request(app)
    .post('/api/products')
    .send(newProduct);
  expect(response.statusCode).toBe(201);
  expect(response.body.name).toBe(newProduct.name);
  expect(response.body.description).toBe(newProduct.description);
});
```

이미 작성 되어 있는
실제 코드

```
exports.createProduct = async (req, res, next) => {
  try {
    const createdProduct = await Product.create(req.body);
    res.status(201).json(createdProduct);
  } catch (err) {
    next(err);
  }
};
```

에러

listen EADDRINUSE: address aleady in use :::5000

이유

supertest로 인해서 이미 서버가 5000번에서 돌아가는데
원래 코드로 인해 다시 한번 서버가 켜지려 하니 이 에러가 납니다

해결 방법

원래 서버가 켜지는 부분을 Commend 처리 해줍니다.

에러 처리를 위한 통합 테스트

해야 할 일은 ???

→ Product 데이터를 데이터베이스에 넣기

통합 테스트 작성

```
it("should return 500 on POST /api/products", async () => {
  const response = await request(app)
    .post('/api/products')
    .send({ name: "Missing description property" });
  expect(response.statusCode).toBe(500);
  expect(response.body).toStrictEqual({
    message: "Product validation failed"
  });
});
```

이미 작성 되어 있는 실제 코드

```
exports.createProduct = async (req, res, next) => {
  try {
    const createdProduct = await Product.create(req.body);
    res.status(201).json(createdProduct);
  } catch (err) {
    next(err);
  }
};
```

설명하는 소스 코드 출처 !
http://thecodebarbarian.com/80-20-guide-to-express-error-handling.html

Express 에러 처리에 대해서

```
const app = require('express')();

app.get('*', function(req, res, next) {
  // This middleware throws an error, so Express will go straight to
  // the next error handler
  throw new Error('woops');
});

app.get('*', function(req, res, next) {
  // This middleware is not an error handler (only 3 arguments),
  // Express will skip it because there was an error in the previous
  // middleware
  console.log('this will not print');
});

app.use(function(error, req, res, next) {
  // Any request to this server will get here, and will send an HTTP
  // response with the error message 'woops'
  res.json({ message: error.message });
});

app.listen(3000);
```

이렇게 이 미들웨어에
에러가 발생을 하면
익스프레스는
이 에러를 에러 처리기 (Handler)로
보내줍니다.

위에 에러가 발생했기 때문에
에러 처리기로 바로 가야 하기
때문에 이 미들웨어는 생략해줍니다.
왜냐하면 이 미들웨어는
에러 처리기(Error Hanlder)가 아니기
때문입니다.

에러 처리기는
이렇게 4개의 인자가 들어갑니다.
그래서 첫번째 미들웨어에서
발생한 에러 메시지를
이곳에서 처리해줍니다.

원래는 위에서 처럼 에러 처리를 해주면 되지만,
비동기 요청으로 인한 에러를 이렇게 처리해주면
에러 처리기에서 저 에러 메시지를 받지 못하기 때문에
서버가 Crash 되어 버립니다.

```
const app = require('express')();

app.get('*', function(req, res, next) {
  // Will crash the server on every HTTP request
  setImmediate(() => { throw new Error('woops'); });
});

app.use(function(error, req, res, next) {
  // Won't get here, because Express doesn't catch the above error
  res.json({ message: error.message });
});
```

```
app.listen(3000);
```

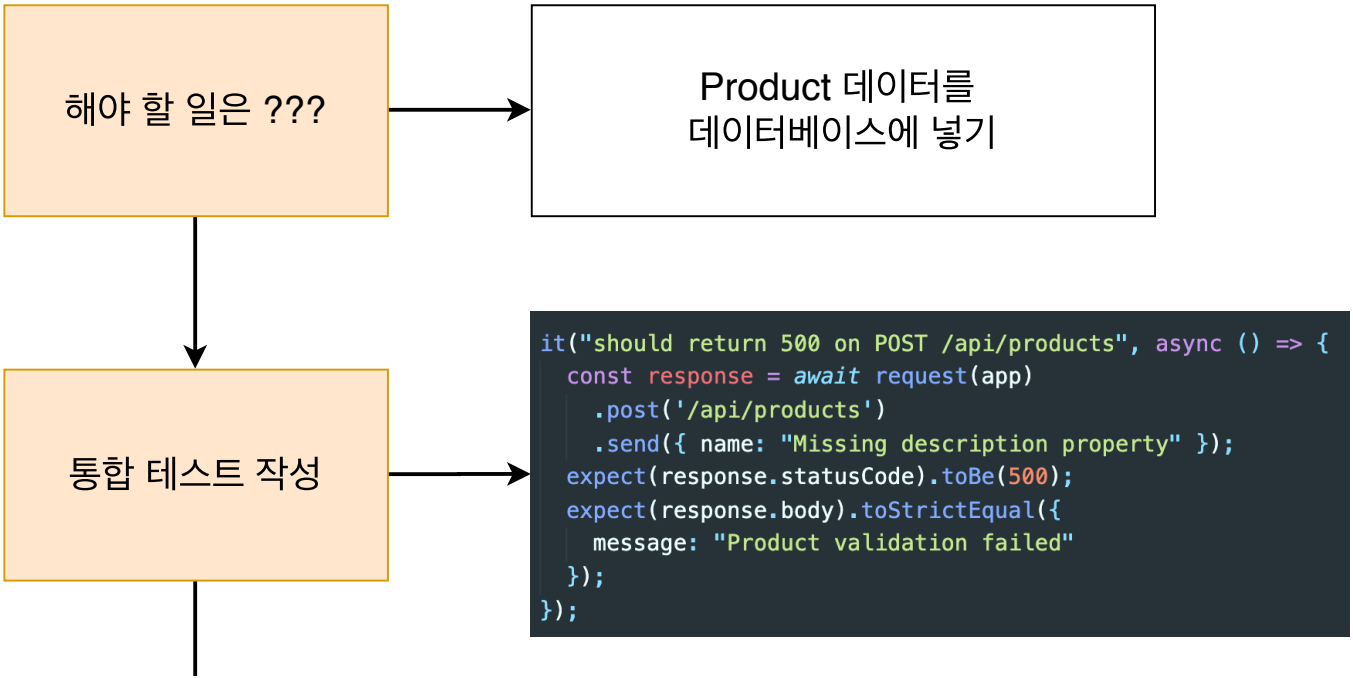
HOW TO SOLVE ?

```
const app = require('express')();

app.get('*', function(req, res, next) {
  // Reporting async errors *must* go through `next()`
  setImmediate(() => { next(new Error('woops')); });
});

app.use(function(error, req, res, next) {
  // Will get here
  res.json({ message: error.message });
});

app.listen(3000);
```




이미 작성 되어 있는
실제 코드
exports.createProduct = async (req, res, next) => {
try {
const createdProduct = await Product.create(req.body);
res.status(201).json(createdProduct);
} catch (err) {
next(err);
}
};