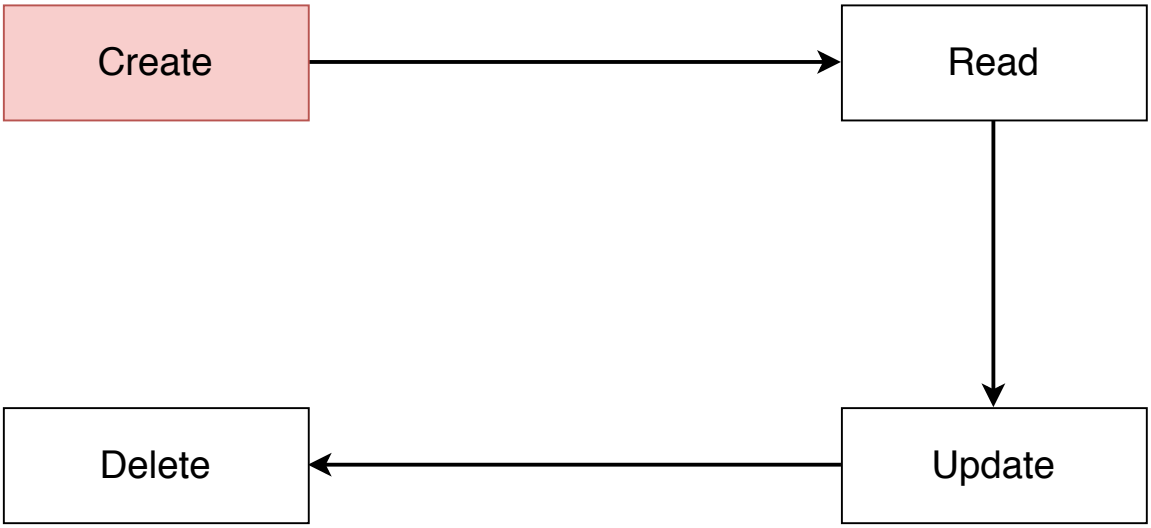
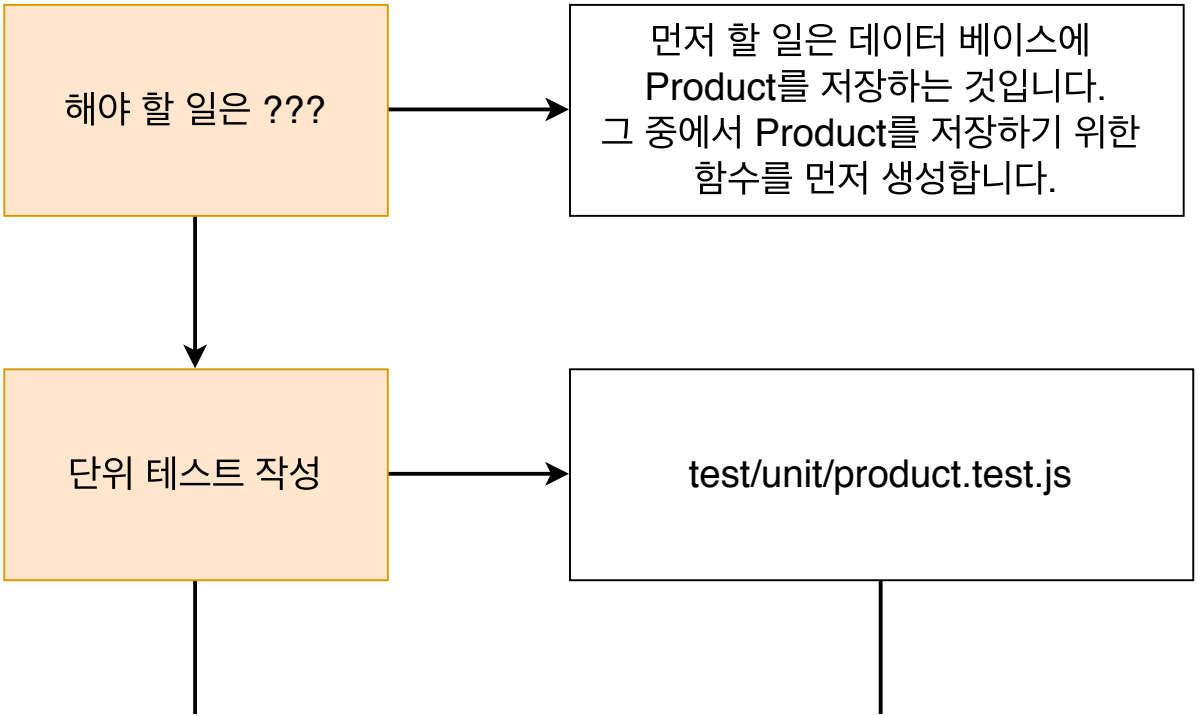
Create 부분 (1) Create Product

CRUD 작성 순서



Create 부분 소스 작성



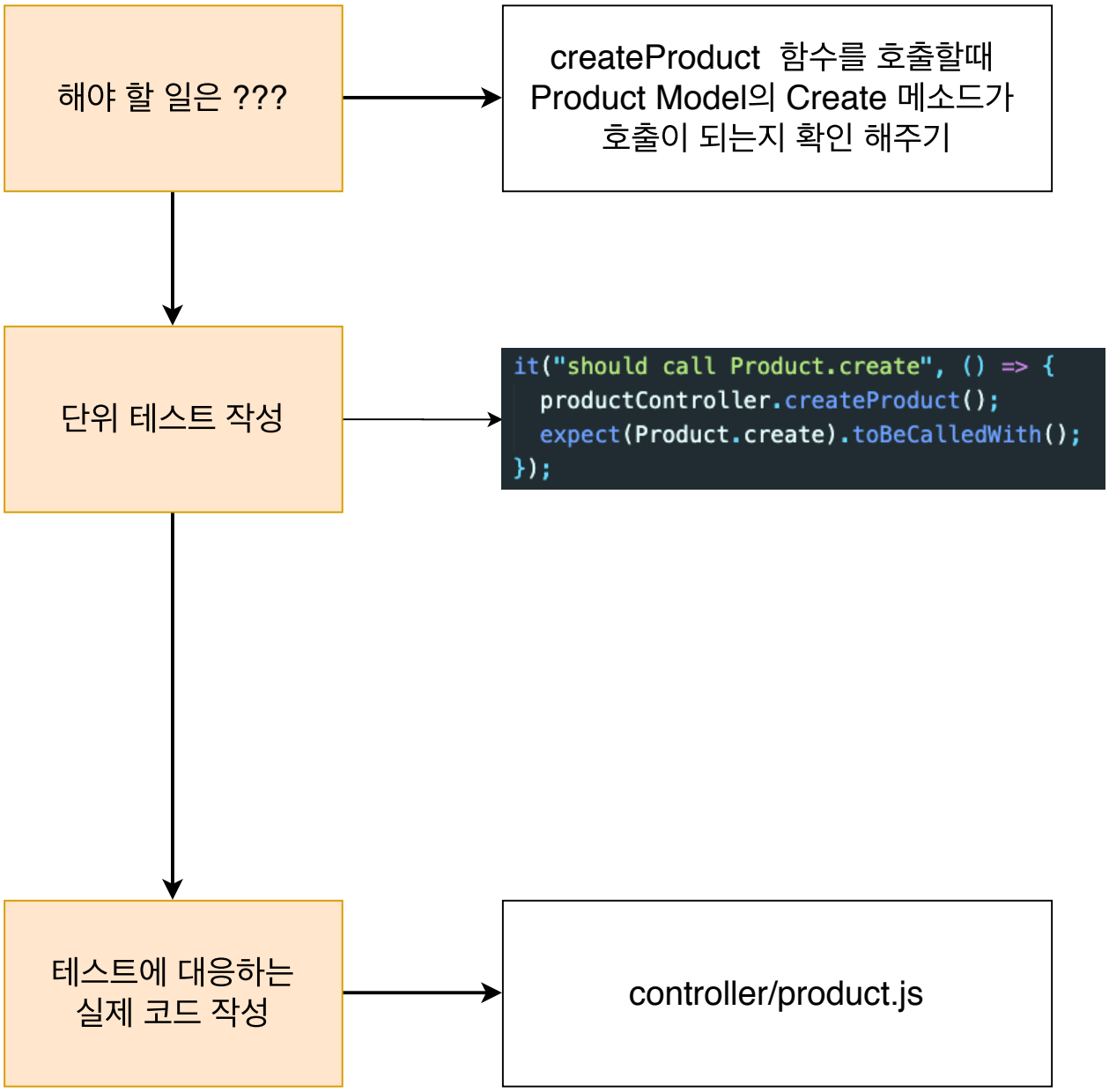
```
describe("Product Controller Create", () => {
  it("should have a createProduct function", () => {
    expect(typeof productController.createProduct).toBe("function");
  });
});
```

테스트에 대응하는
실제 코드 작성

controller/product.js

```
exports.createProduct = () => {};
```

Create 부분 (2) Create Method



몽구스를 사용할 때 나오는 경고 메시지

```
console.warn Mongoose: looks like you're trying to test a Mongoose app with Jest's default jsdom test environment.
Please make sure you read Mongoose's docs on configuring Jest to test Node.js apps: http://mongoosejs.com/docs/jest.html
```

Node.js 앱을 테스트하기 위한 Jest 설정

몽구스를 사용할 때 나오는 경고 메시지

```
console.warn Mongoose: looks like you're trying to test a Mongoose app with
Jest's default jsdom test environment.
Please make sure you read Mongoose's docs on configuring Jest to test
Node.js apps: http://mongoosejs.com/docs/jest.html
```

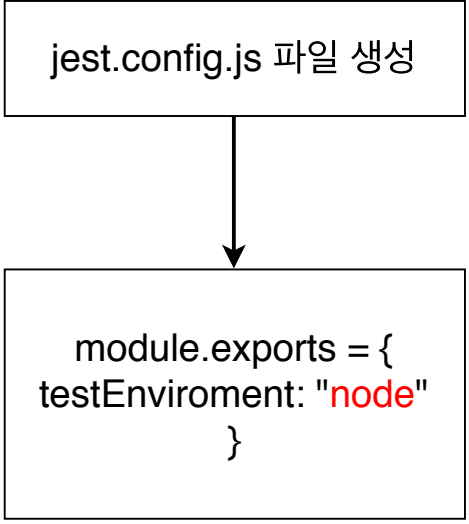
해결하기 위해서는...

문제점은....

해결책은...

Jest의 기본 Test 환경

jsdom ==========> node


jest.config.js 파일 생성
module.exports = {
testEnviroment: "node"
}

Create 부분 (3) node-mocks-http

해야 할 일은 ???

Product.create() 아직 저장할 Product 데이터를 넣어 주지 않았습니다.
그래서 이제는 데이터베이스에 저장할 데이터를 넣어주겠습니다.

단위 테스트 작성

원래 몽구스 모델을 이용해서 데이터를 저장할때는

Product.create() ====> Product.create(req.body)
이런식으로 req 객체를 이용해서 요청에 함께 들어온 body를 create메소드에 인자로 넣어줘서 데이터베이스에 저장해줍니다.
그래서 단위 테스트에서도 req 객체가 필요합니다.
그러면 어떻게 이 req 객체를 단위 테스트에서 이용할까요 ????

단위 테스트에서 reqest, response 객체를 얻으려면???

node-mocks-http 모듈을 이용!!!

node-mocks-http 모듈을 이용해서 Express.js 애플리케이션 라우팅 함수를 테스트하기 위한 http 객체(request, response) 를 얻는 방법

```
req = httpMocks.createRequest();
```

```
res = httpMocks.createResponse();
```

req 객체를 얻었다면
req.body 안에다가 저장해줄
Product를 넣어주기

```
req.body = newProduct;
productController.createProduct(req, res, next);
```

expect와 matcher를 통해서
데이터베이스에 데이터가 되는
부분 코드 테스트 해보기

```
req.body = newProduct;
productController.createProduct(req, res, next);
expect(Product.create).toBeCalledWith(newProduct);
```

테스트에 대응하는
실제 코드 작성

controller/product.js

```
exports.createProduct =  (req, res, next) => {
    Product.create(req.body);
};
```

beforeEach 사용하기

beforeEach

여러 개의 테스트에 공통된 Code가 있다면
beforeEach 안에 넣어서 반복을 줄여줄 수 있습니다.

Jest 파일 구조

beforeEach

describe

test (it)

test (it)

describe

test (it)

test (it)

```
beforeEach(() => {
    req = httpMocks.createRequest();
    res = httpMocks.createResponse();
    next = null;
});
```

Create 부분 (4) 상태값 전달

해야 할 일은 ???

데이터베이스에 데이터를 저장했으니
이제 그 결과값을
클라이언트에 전달 해줘야 합니다.
상태 결과 값을 보내줍니다.

단위 테스트 작성

성공적으로 데이터를 Create 하면
201 Status를 Response로 보냅니다.

```
it("should return 201 response code", () => {
  productController.createProduct(req, res, next);
  expect(res.statusCode).toBe(201);
});
```

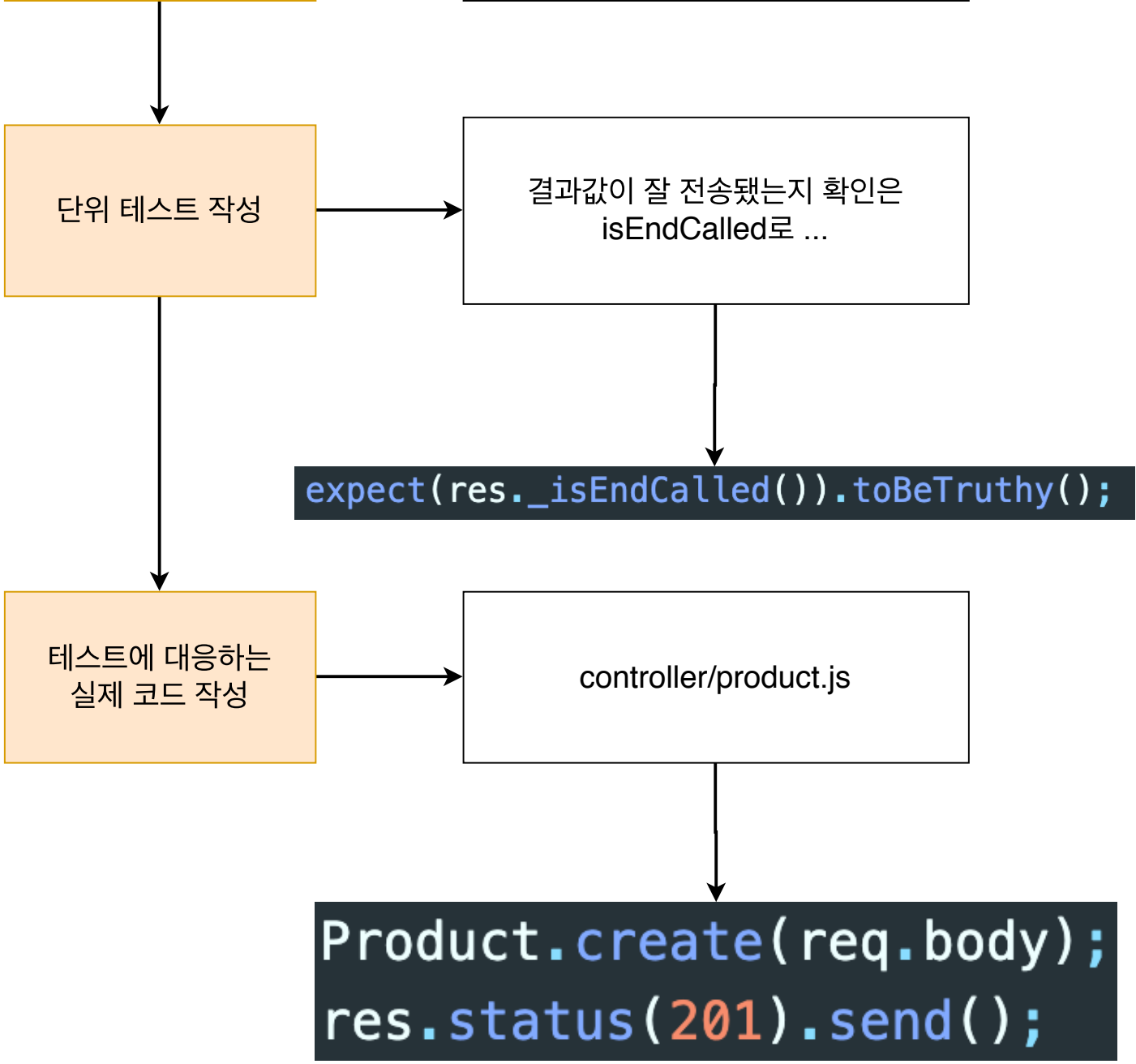
테스트에 대응하는
실제 코드 작성

controller/product.js

```
Product.create(req.body);
res.status(201)
```

해야 할 일은 ???

결과값이 잘 전송됐는지 확인


단위 테스트 작성
결과값이 잘 전송됐는지 확인은
isEndCalled로 ...
expect(res._isEndCalled()).toBeTruthy();
테스트에 대응하는
실제 코드 작성
controller/product.js
Product.create(req.body);
res.status(201).send();

Create 부분 (5) Response results back

해야 할 일은 ???

새로 저장된(생성된) 데이터를
결과값으로 보내줍니다.

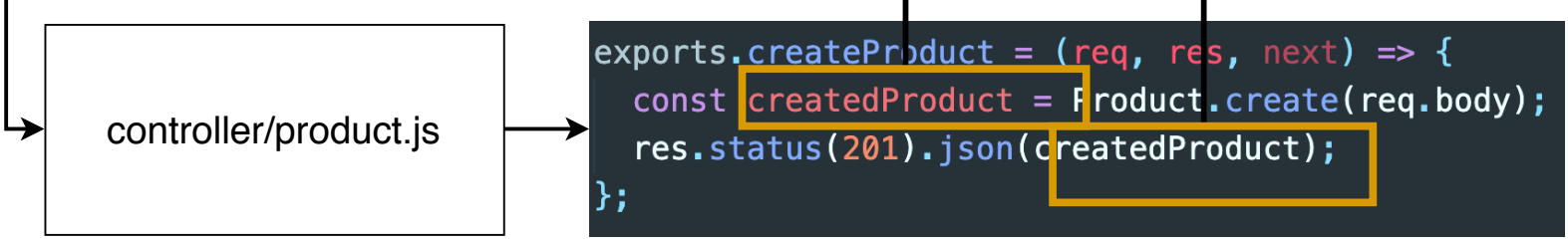
```
const createdProduct = Product.create(req.body);
```

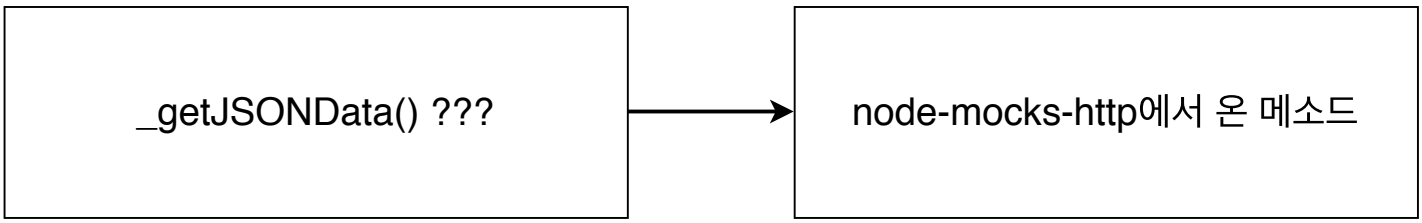
단위 테스트 작성

테스트에 대응하는
실제 코드 작성

가짜 함수가
어떠한 결과값(Return)을
반환 할지 직접 알려줄때는
mockReturnValue를 사용합니다.

```
it("should return json body in response", () => {
  Product.create.mockReturnValue(newProduct);
  productController.createProduct(req, res, next);
  expect(res._getJSONData()).toStrictEqual(newProduct);
});
```





`.toStrictEqual(value)`

Use `.toStrictEqual` to test that objects have the same types as well as structure.

Differences from `.toEqual`:

- Keys with `undefined` properties are checked. e.g. `{a: undefined, b: 2}` does not match `{b: 2}` when using `.toStrictEqual`.
- Array sparseness is checked. e.g. `[, 1]` does not match `[undefined, 1]` when using `.toStrictEqual`.
- Object types are checked to be equal. e.g. A class instance with fields `a` and `b` will not equal a literal object with fields `a` and `b`.

```
class LaCroix {
  constructor(flavor) {
    this.flavor = flavor;
  }
}

describe('the La Croix cans on my desk', () => {
  test('are not semantically the same', () => {
    expect(new LaCroix('lemon')).toEqual({flavor: 'lemon'});
    expect(new LaCroix('lemon')).not.toStrictEqual({flavor: 'lemon'});
  });
});
```