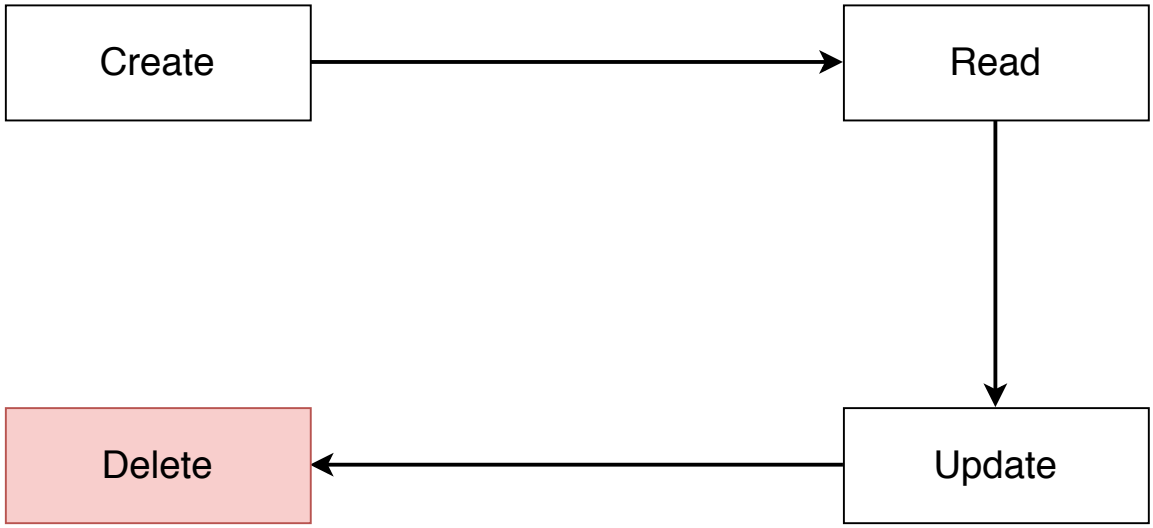
DELETE 시작

CRUD 작성 순서



Delete 부분 소스 작성

해야 할 일은 ???

먼저 할 일은 데이터 베이스에
있는 특정 Product 데이터를
삭제 하는것 입니다.
그래서 특정 Product 데이터를
삭제 하기 위한 함수를 먼저 생성합니다.

단위 테스트 작성

```
describe("Product Controller Delete", () => {
  Run | Debug
  it("should have a deleteProduct function", () => {
    expect(typeof productController.deleteProduct).toBe("function");
  });
});
```

테스트에 대응하는
실제 코드 작성

```
exports.deleteProduct = async (req, res, next) => {};
```

deleteProduct 단위 테스트 작성

Delete 부분 소스 작성

해야 할 일은 ???

→ deleteProduct 함수를 호출할 때 Product Model의 findByIdAndDelete 메소드 호출

단위 테스트 작성

```
it("should call ProductModel.findByIdAndDelete", async () => {
  req.params.productId = productId;
  await productController.deleteProduct(req, res, next);
  expect(Product.findByIdAndDelete).toBeCalledWith(productId);
});
```

테스트에 대응하는 실제 코드 작성

```
exports.deleteProduct = async (req, res, next) => {
  await Product.findByIdAndDelete(req.params.productId);
};
```

해야 할 일은 ???

→ 데이터 삭제 성공 후 성공 status와 deleted된 데이터를 클라이언트로 전송하기

```
it("should return 200 response", async () => {
  productModel.findByIdAndDelete.mockReturnValue(newProduct);
```

단위 테스트 작성

```
  await productController.deleteProduct(req, res, next);
  expect(res.statusCode).toBe(200);
  expect(res._getJSONData()).toStrictEqual(newProduct);
  expect(res._isEndCalled()).toBeTruthy();
});
```

테스트에 대응하는
실제 코드 작성

```
exports.deleteProduct = async (req, res, next) => {
  const deletedProduct = await Product.findByIdAndDelete(req.params.productId);
  res.status(200).json(deletedProduct);
};
```

해야 할 일은 ???

삭제하는 데이터를
찾지 못한 경우

단위 테스트 작성

```
it("should handle 404 when item doesnt exist", async () => {
  productModel.findByIdAndDelete.mockReturnValue(null);
  await productController.deleteProduct(req, res, next);
  expect(res.statusCode).toBe(404);
  expect(res._isEndCalled()).toBeTruthy();
});
```

테스트에 대응하는
실제 코드 작성

```
exports.deleteProduct = async (req, res, next) => {
  const deletedProduct = await Product.findByIdAndDelete(req.params.productId);
  if (deletedProduct) {
    res.status(200).json(deletedProduct);
  } else {
    res.status(404).send();
  }
};
```

해야 할 일은 ???

데이터를 삭제할 때 에러가 난 경우

단위 테스트 작성

```
it("should handle errors", async () => {
  const errorMessage = { message: "Error deleting" };
  const rejectedPromise = Promise.reject(errorMessage);
  productModel.findByIdAndDelete.mockReturnValue(rejectedPromise);
  await productController.deleteProduct(req, res, next);
  expect(next).toHaveBeenCalledWith(errorMessage);
});
```

테스트에 대응하는
실제 코드 작성

```
exports.deleteProduct = async (req, res, next) => {
  try {
    const deletedProduct = await Product.findByIdAndDelete(req.params.productId);
    if (deletedProduct) {
      res.status(200).json(deletedProduct);
    } else {
      res.status(404).send();
    }
  } catch (err) {
    next(err);
  }
};
```

deleteProduct 통합 테스트 작성

Delete 부분 소스 작성

해야 할 일은 ???

특정 데이터를 삭제 하기

통합 테스트 작성

```
test("DELETE /api/products", async () => {
  const res = await request(app)
    .delete("/api/products" + newProductId)
    .send();
  expect(res.statusCode).toBe(200);
});
```

테스트에 대응하는 실제 코드 작성

```
exports.deleteProduct = async (req, res, next) => {
  try {
    const deletedProduct = await productModel.findByIdAndDelete(req.params.productId);
    if (deletedProduct) {
      res.status(200).json(deletedProduct);
    } else {
      res.status(404).send();
    }
  } catch (err) {
    next(err);
  }
};
```

해야 할 일은 ???

특정 데이터가 데이터베이스에 없을 때

통합 테스트 작성

```
test("DELETE id doenst exist /api/products/:productId", async () => {
  const res = await request(app)
    .delete("/api/products/" + nonExistingProductId)
    .send();
  expect(res.statusCode).toBe(404);
});
```

테스트에 대응하는
실제 코드 작성

```
exports.deleteProduct = async (req, res, next) => {
  try {
    const deletedProduct = await productModel.findByIdAndDelete(req.params.productId);
    if (deletedProduct) {
      res.status(200).json(deletedProduct);
    } else {
      res.status(404).send();
    }
  } catch (err) {
    next(err);
  }
};
```