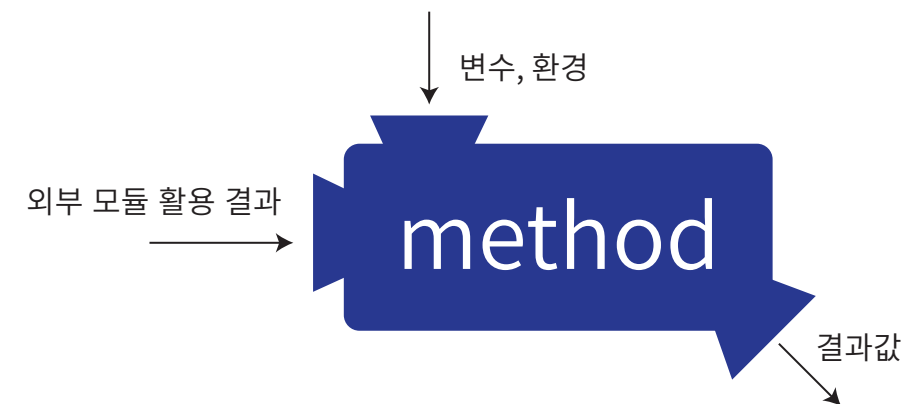


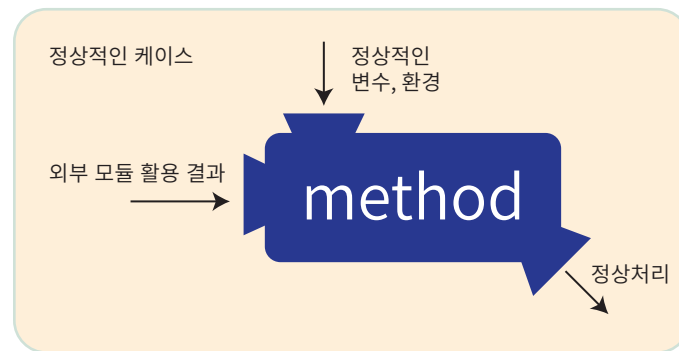
users.controller.js

UserController 클래스가 있다.

그 중에서도 한 메소드의 기능을 어떻게 테스트할까?

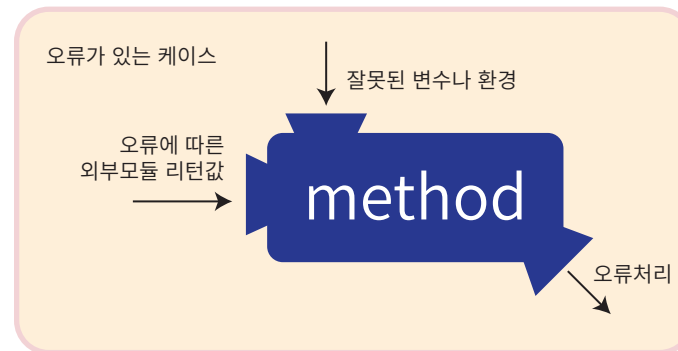


여러 시나리오(Case)에서 이 단위 자체가 독립적으로 잘 구동하려면 케이스별로 이를 둘러싼 인풋(변수, 환경, 외부모듈 활용한 결과값)을 통제할 수 있어야 한다.



ex. 가짜 request를 보낼 수 있다.

super-test



ex. 가짜 res, req, next 객체를 만들 수 있다.

node-mocks-http

ex. jest.fn()을 통해 가짜 함수로 만들어 메소드 중간에 리턴할 값을 미리 통제할 수 있다.

jest

새로 install 한 이 모듈들은 테스트를 위한 이런 가짜 환경, 즉 mock환경을 만들어내는 도구들이 된다.

테스트코드는 이런식이다.

```
Run | Debug
it("기능: 사용자가 지금까지 좋아한 리스트에 이번 게시글이 없는 경우, ~~ 이래야 한다.", async () => {
  // 전제된 경우, Repository들이 아래값을 리턴한다고 치자.
  postsService.postRepository.getPost.mockReturnValue(
    postDataOut.getPostDetailRes.data
  );
  postsService.userRepository.getAllLikedPosts.mockReturnValue(["1", "3"]); // 기존에 좋아한 배열이 1,3인 상태에서

  // 로그인한 유저가
  await postsService.likePost(res.locals.user, "4"); // 4번을 좋아하면

  // post저장소와 user저장소의 좋아요(likePost) 메소드가 각각 1번씩 호출되어야 한다.
  expect(postsService.postRepository.likePost.mock.calls.length).toBe(1);
  expect(postsService.userRepository.likePost.mock.calls.length).toBe(1);
});
```

단위 테스트 코드는 각 부품이 해결해야할 상황을 디자인하고 통제해서 돌려보는 것이다.

