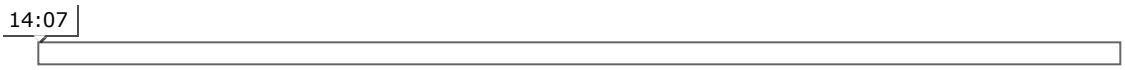
Watch: 🗣️ **Entendiendo cuándo es necesario mockear**



Puedes encontrar el código de este vídeo en GitHub.

Anteriormente en el curso hemos visto que el punto de corte de nuestros tests unitarios sería el servicio que conecta con algo externo de nuestra aplicación, como el repositorio que hace `fetch` a una API.

Jest nos permite auto-mockear cualquier módulo añadiendo esta línea en nuestro archivo de test:

```
jest.mock("../services/CoursesRepository");
```

Esto transforma los exports de ese archivo a funciones mock. Esto nos permite despreocuparnos de no llamar a la api real, y en cada test

podemos ajustar el mock para que devuelva lo que queramos usando `.mockResolvedValue`

```
coursesRepo.searchAll.mockResolvedValue([{
  slug: "my-course",
  title: "My course title",
  img: "img.jpg",
  description: "Lorem ipsum",
}]);
```

También podemos usar `mockImplementation` para pasar una implementación custom, por ejemplo en el caso que queramos devolver algo pasado por parámetro:

```
createUser.mockImplementation(async (data) => await {
  id: 1,
  ...data
});
```

Para asegurar que no arrastramos valores no esperados a otros tests, es buena idea limpiar el mock después de cada test. Podemos añadirlo manualmente, usar `afterEach`, o añadirlo en el archivo `jest.config.js`

```
module.exports = {
  testMatch: ["**/*.test.js"],
  // ...
  restoreMocks: true,
};
```

En el test del ejemplo vemos que mockeamos el repositorio que contacta con la api, pero tenemos otro servicio que no mockeamos, `timeService`. ¿Por qué?

Ese servicio usa una librería externa, DayJs. Es un paquete que instalamos y no va a tener un gran impacto en el tiempo de ejecución de nuestros tests, ya que no tiene que contactar con una api via http. Si decidimos mockearla, tendríamos que crear otro test de integración para asegurar que nuestro código funciona bien con la librería, ya que si

la actualizamos y se rompe algo necesitamos saberlo. Debido al coste bajo que tiene testearlo sin mockear nada en el unit test, en este caso consideramos que tiene sentido prescindir del test de integración y no doblar ese servicio, siguiendo la filosofía de Testing Library de mockear lo menos posible y testear lo más parecido a como se usa el software final.

Como siempre, es una decisión que hay que tomar según las circunstancias de cada caso y según lo que decida el equipo.

Links relacionados:

- Testing Library: Guiding Principles
- Jest: Mock Functions
- Jest: Manual Mocks guide