What Are the Differences Between Mocking, Faking, and Stubbing?

Mocking is the process of replacing real data with false data that simulates the behavior of real components. This allows testers and developers to test for specific components in the application without having to rely on external dependencies.

A common use case for mocking data would be when dealing with API calls. Imagine you have a fetch call like this in your application:

```
export const postService = {
  async getPosts() {
    const response = await fetch("https://api.example.com/posts");
    return response.json();
  },
};
```

If you need to test the `postService` function, you could mock the API response instead of making continuous API calls to fetch the data. Fetching from an API takes time and depends on external factors like network availability and server responses. Mocking the API response removes those external factors and allows testers to work with a predictable, controlled data set.

Here is an example mock API response:

```
[
  {
    "id": 1,
    "title": "Understanding Async/Await in JavaScript",
    "content": "Async/Await makes asynchronous code look and behave like synchronous
code...",
    "author": "Jane Doe",
    "created_at": "2025-04-10T14:32:00Z"
  },
  {
    "id": 2,
    "title": "10 Tips for Writing Clean Code",
    "content": "Clean code is not just about code formatting. It involves naming,
architecture...",
    "author": "John Smith",
    "created_at": "2025-04-08T09:21:00Z"
  },
  {
    "id": 3,
    "title": "Exploring React 19 Features",
    "content": "React 19 comes with exciting features like new hooks and better
performance...",
    "author": "Alex Lee",
    "created_at": "2025-04-07T18:47:00Z"
  }
]
```

Now that we understand what mocking is, let's compare it with stubbing. Stubs are objects that return pre-defined responses or dummy data for an expected behavior in an application. For example, you can stub the behavior for a database connection in your tests without having to rely on an actual database connection.

Here is an example of a stub in Jest:

```
import { postService } from "./postService";
import { db } from "./db";

jest.mock("./db", () => ({
  db: {
    fetchPosts: jest.fn(),
  },
}));

test("should return stubbed posts", async () => {
  const fakePosts = [
    { id: 1, title: "Stubbed Post", content: "This is a stubbed post." },
  ];

  db.fetchPosts.mockResolvedValue(fakePosts);

  const posts = await postService.getPosts();

  expect(posts).toEqual(fakePosts);
});
```

`db.fetchPosts.mockResolvedValue(fakePosts);` is the stub in this example.

The last component to discuss are fakes. Fakes are simplified versions of real components without the complexity or side effects of the real thing. For example, you can fake a database by storing the data in memory instead of interacting with the real database. This will allow you to mimic database operations in memory, which will be much faster than dealing with the real database. Another common example would be to use fakes when working with file systems. You can create a fake file system in place of the real one when it comes to testing.

As you continue to build out more complex software applications, consider using mocks, stubs, or fakes where appropriate in your testing.

Questions

Which of the following is a common use case to use a mock in testing?

Mocking HTML changes.

Mocking CSS changes.

Mocking API calls.

Correct!

Mocking JavaScript string methods.

What are stubs in testing?

Objects that return null in testing environments.

Objects that return pre-defined responses.

Correct!

Objects that make multiple API calls.

Objects that make several updates to the UI in React applications.

Which of the following is a common use case for a fake?

Faking API calls for post requests.

Faking the styles for an application.

Faking an HTML file.

Faking a database.

Correct!