Take 2

API Testing

API Testing with Jest

- Why test APIs?
- Ensuring endpoints work as expected
- Catching regressions before production
- Automating with CI/CD

Excursion: Deployment & CI/CD

- "Deploying" means running your new code live, e.g. on the internet
- Modern teams often deploy automatically when a dev pushes to git
  - This is called **Continuous Deployment** (CD)
- Before auto-deployment, *tests* are run to ensure the code about to go live works as expected
  - This is called **Continuous Integration** (CI)

Jest: Quick Intro

- A JavaScript testing framework
- Offers test runner, assertions, and more
- Zero config (in most cases)
- Three main functions: `describe`, `test` (or `it`), `expect`

Setup & Installation

When using `npm`, install jest as a *dev* dependency

```
$ npm install --save-dev jest
```

- No need to do anything when using `bun`, it integrates the jest functionality
- Add `"test": "jest"` to scripts in `package.json`
- Optionally configure with `jest.config.js`

Basic Test Example

```
// sum.js
export function sum(a, b) {
  return a + b;
}
```

```
// sum.test.js
import { sum } from "./sum";

test("adds 1 + 2 to equal 3", () => {
  expect(sum(1, 2)).toBe(3);
});
```

- Run `npm test` / `bun test`

Testing Express APIs

- Start server
- Use `fetch` to call local endpoints
- Check response statuses, bodies

```
// server.js
import express from "express";
const app = express();

app.get("/api/hello", (req, res) => {
  res.json({ message: "Hello from API" });
});

export default app;
```

Example API Test

```
// server.test.js
describe("GET /api/hello", () => {
  it("should return the greeting message", async () => {
    const response = await fetch(`http://localhost:3000/api/hello`);
    expect(response.status).toBe(200);
    const json = await response.json();
    expect(json.message).toBe("Hello from API");
  });
});
```

Coverage & Reports

- `--coverage` flag
- Generate coverage reports (lcov, text, HTML)
- Helps identify untested code paths

```
$ npm test -- --coverage
$ bun test --coverage
```

Database Fixtures

- **What are fixtures?** Predefined data sets used during tests
- **Why use them?**
  - Ensure predictable database state
  - Speed up test setup and teardown
  - Promote reproducible and isolated tests

Example: Loading a Fixture

```
// test.js

test("GET /api/users returns fixture data", async () => {
  await User.insertMany([{ name: "Frodo" }, { title: "Bilbo" }]);
  ...
});
```

- Ensure tests start from a known data state
- Consider teardown or truncation after each test (see `api.test.js`)