0. Big picture (what we're trying to do)

For Cypress tests, we want this flow:

1. **Backend has a separate test DB** (so we don't break real data).
2. **Seed script** in Node that:
    - Clears tables
    - Inserts known test records (from a JSON fixture).
3. **Cypress custom command** that:
    - Calls a special backend endpoint (or task) to run the seed script.
4. **Tests use that command** in `beforeEach()` so every test runs against a **clean, predictable DB**.

---

1. Setup a simple Express + SQLite project

If you already have an app, you can just map the relevant parts. Otherwise:

```
mkdir cypress-db-seeding-demo
cd cypress-db-seeding-demo
npm init -y
npm install express sqlite3
npm install --save-dev nodemon cypress
```

Create a basic folder structure:

```
cypress-db-seeding-demo
├─ src
│  ├─ db.js
│  ├─ server.js
│  └─ seed.js
├─ cypress
│  ├─ e2e
│  ├─ fixtures
```

```
│   └── support
│       └── e2e.js
└── package.json
```

---

2. Create a small SQLite DB (dev + test)

`src/db.js`

Here we create **two DB files**: `dev.db` and `test.db`, based on `NODE_ENV`.

```
// src/db.js
const sqlite3 = require('sqlite3').verbose();
const path = require('path');

const isTest = process.env.NODE_ENV === 'test';
const dbFile = isTest ? 'test.db' : 'dev.db';

const db = new sqlite3.Database(path.join(__dirname, '..', dbFile));

// Simple helper to run schema creation once
db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL,
      email TEXT UNIQUE NOT NULL
    )
  `);
});

module.exports = db;
```

This gives you a **very small DB** with a `users` table.

---

3. Build a simple API for users

`src/server.js`

```
// src/server.js
const express = require('express');
const bodyParser = require('body-parser');
const db = require('./db');

const app = express();
app.use(bodyParser.json());

// Get all users
app.get('/api/users', (req, res) => {
  db.all('SELECT * FROM users', [], (err, rows) => {
    if (err) return res.status(500).json({ error: err.message });
    res.json(rows);
  });
});

// Create user
app.post('/api/users', (req, res) => {
  const { name, email } = req.body;
  db.run(
    'INSERT INTO users (name, email) VALUES (?, ?)',
    [name, email],
    function (err) {
      if (err) return res.status(500).json({ error: err.message });
      res.status(201).json({ id: this.lastID, name, email });
    }
  );
});

// Start server only if run directly (not when required by tests)
if (require.main === module) {
  const port = process.env.PORT || 4000;
  app.listen(port, () => console.log(`Server running on port ${port}`));
}

module.exports = app;
```

Update `package.json` scripts to run dev and test servers:

```
"scripts": {
  "dev": "nodemon src/server.js",
  "start:test": "NODE_ENV=test node src/server.js",
  "cypress:open": "cypress open",
```

```
  "cypress:run": "cypress run"
}
```

---

4. Create a seeding script in Node

`src/seed.js`

This will:

- **Delete all rows** from `users`
- **Insert known data** (we'll plug fixtures soon)

For now, hardcode a few users:

```
// src/seed.js
const db = require('./db');

function seedUsers(users) {
  return new Promise((resolve, reject) => {
    db.serialize(() => {
      db.run('DELETE FROM users', (err) => {
        if (err) return reject(err);

        const stmt = db.prepare('INSERT INTO users (name, email) VALUES (?, ?)');
        users.forEach((u) => {
          stmt.run(u.name, u.email);
        });
        stmt.finalize((err2) => {
          if (err2) return reject(err2);
          resolve();
        });
      });
    });
  });
}

// When run directly from Node CLI
if (require.main === module) {
  // Example hardcoded data if you want to run: node src/seed.js
```

```
  const exampleUsers = [
    { name: 'Alice', email: 'alice@test.com' },
    { name: 'Bob', email: 'bob@test.com' }
  ];

  seedUsers(exampleUsers)
    .then(() => {
      console.log('Seed complete');
      process.exit(0);
    })
    .catch((err) => {
      console.error('Seed failed', err);
      process.exit(1);
    });
}

module.exports = { seedUsers };
```

---

5. Expose a test-only seed endpoint for Cypress

We'll create an endpoint `/test/seed` that Cypress can call.
It will:

- Read users from the request body
- Call our `seedUsers` function

Update `src/server.js`:

```
const { seedUsers } = require('./seed');

// ... existing code above

// Test-only endpoint (only in test NODE_ENV ideally)
if (process.env.NODE_ENV === 'test') {
  app.post('/test/seed', async (req, res) => {
    try {
      const { users } = req.body;
      await seedUsers(users);
      res.json({ ok: true });
    } catch (err) {
```

```
      console.error(err);
      res.status(500).json({ error: err.message });
    }
  });
}
```

In a real project, you'd protect this endpoint and only enable it in test env.

---

6. Create Cypress fixtures (test data as JSON)

Create file: `cypress/fixtures/users.json`

```
[
  { "name": "Test User 1", "email": "test1@example.com" },
  { "name": "Test User 2", "email": "test2@example.com" },
  { "name": "Test User 3", "email": "test3@example.com" }
]
```

---

7. Create a custom Cypress command for seeding

We'll add a custom command `cy.seedDatabase()` that:

1. Loads `users.json` with `cy.fixture`
2. Calls the backend `/test/seed` API with those users

Edit `cypress/support/e2e.js`:

```
// cypress/support/e2e.js

Cypress.Commands.add('seedDatabase', () => {
  cy.fixture('users').then((users) => {
    cy.request('POST', 'http://localhost:4000/test/seed', { users });
  });
});
```

TypeScript note: if TS is enabled you'd also add a `cypress/support/index.d.ts` for typings, but we'll skip that for now.

---

8. Use seeding in a test

Create a test file: `cypress/e2e/users.cy.js`

```
// cypress/e2e/users.cy.js

describe('Users API with DB seeding', () => {
  // Before each test, reset + seed DB
  beforeEach(() => {
    cy.seedDatabase();
  });

  it('returns seeded users from the API', () => {
    cy.request('GET', 'http://localhost:4000/api/users').then((response) => {
      expect(response.status).to.eq(200);
      expect(response.body).to.have.length(3);
      expect(response.body[0]).to.have.property('email', 'test1@example.com');
    });
  });

  it('can create a new user on top of seeded data', () => {
    cy.request('POST', 'http://localhost:4000/api/users', {
      name: 'New User',
      email: 'newuser@example.com'
    }).then((response) => {
      expect(response.status).to.eq(201);
    });

    cy.request('GET', 'http://localhost:4000/api/users').then((response) => {
      expect(response.body).to.have.length(4);
    });
  });
});
```

Run it:

Start the server in **test mode**:

```
 npm run start:test
# server on http://localhost:4000
```

1.

In another terminal run Cypress:

```
 npx cypress open
# or
npm run cypress:run
```

2.

---

9. Best practices for maintaining test data

Here are the key habits you want (especially when teaching others later 😄):

1. Separate databases per environment

- `dev.db`, `test.db`, `prod.db` (or separate schemas).
- Never run Cypress tests against prod data.

2. Reset data for every test (or every suite)

- Use `beforeEach(() => cy.seedDatabase())` for fully isolated tests.
- For heavier setups, you might seed **once per file** in `before()` and then only reset the parts that change.

3. Use fixtures as your “single source of truth”

- Keep all static test data in `cypress/fixtures/*`.
- Use meaningful names: `users_basic.json`, `users_admin.json`.
- This makes test data **visible and version-controlled**.

4. Make seeding fast & deterministic

- Delete and insert, don't do complex logic.
- Same input JSON → always same DB state.
- Avoid randomness inside seeds unless you control it with fixed IDs / emails.

5. Avoid test inter-dependence

Bad:

- Test A creates user, Test B expects that user to exist.

Good:

- Both tests call `cy.seedDatabase()` and know exactly what's in the DB.

6. Keep test-only endpoints behind NODE_ENV

- Never expose `/test/seed` in production.
- Wrap in `if (process.env.NODE_ENV === 'test') { ... }`
- In more advanced setups, use **Cypress `task`** to run DB scripts directly instead of HTTP endpoints.