Schemock Quickstart Guide

Version 1.0.0

Table of Contents

1. Introduction

Welcome to the **Schemock Quickstart Guide**. Schemock is a lightweight, powerful tool designed to generate mock REST APIs directly from JSON schemas.

Purpose:
This guide aims to get you up and running with Schemock in minutes, enabling you to prototype front-end applications, test API integrations, and validate data structures without waiting for backend implementation.

Scope:
We will cover everything from installation to advanced schema configurations, providing you with the knowledge to simulate complex API behaviors including validation, relationships, and error handling.

2. Installation & Setup

Prerequisites

- **Node.js** (v14 or higher)
- **npm** (v6 or higher)

Installation

You can install Schemock globally or as a project dependency.

Global Installation:

```
npm install -g schemock
```

Local Project Installation:

```
npm install schemock --save-dev
```

Verifying Installation

Run the following command to verify the installation:

```
schemock --version
```

---

3. Step-by-Step Implementation

Step 1: Initialize a Project

Create a new directory for your mock server project.

```
mkdir my-mock-server
cd my-mock-server
npm init -y
```

Step 2: Create a Schema File

Create a file named `user-schema.json` in your project root.

```
{
  "name": "users",
  "endpoints": [
    {
      "method": "GET",
      "path": "/users",
      "response": [
        { "id": 1, "name": "John Doe" },
        { "id": 2, "name": "Jane Smith" }
      ]
    }
  ]
}
```

Step 3: Start the Server

Run Schemock pointing to your schema file.

```
schemock --schema user-schema.json --port 3000
```

Step 4: Test the API

Open your browser or use curl to test the endpoint:

```
curl http://localhost:3000/users
```

---

4. Example Schemas

Basic Schema Structure

This example demonstrates the minimal configuration required to define a resource.

File: `basic-product.json`

```
{
  "name": "products",
  "basePath": "/api/v1",
  "endpoints": [
    {
      "method": "GET",
      "path": "/products",
      "response": {
        "data": [
          { "id": "p1", "name": "Laptop", "price": 999 },
          { "id": "p2", "name": "Mouse", "price": 25 }
        ]
      }
    },
    {
      "method": "GET",
      "path": "/products/:id",
      "response": {
        "id": "p1",
        "name": "Laptop",
        "price": 999
      }
    }
  ]
}
```

Advanced Configuration

Use dynamic responses, delays, and status codes to simulate real network conditions.

File: `advanced-config.json`

```json
{
  "name": "orders",
  "delay": 500,
  "endpoints": [
    {
      "method": "POST",
      "path": "/orders",
      "statusCode": 201,
      "response": {
        "message": "Order created successfully",
        "orderId": "ord-{{timestamp}}"
      }
    },
    {
      "method": "GET",
      "path": "/orders/error",
      "statusCode": 500,
      "response": {
        "error": "Internal Server Error",
        "code": "ERR_500"
      }
    }
  ]
}
```

Real-World Scenario: E-commerce API

A comprehensive example simulating a user cart workflow.

File: `ecommerce-cart.json`

```
{
  "name": "cart",
  "basePath": "/api/shop",
  "endpoints": [
    {
      "method": "GET",
      "path": "/cart",
      "response": {
        "items": [
          { "productId": 101, "qty": 2 },
          { "productId": 205, "qty": 1 }
        ],
        "total": 150.00
      }
    },
    {
      "method": "POST",
      "path": "/cart/checkout",
      "statusCode": 200,
      "response": {
        "status": "success",
        "transactionId": "tx-789012",
        "estimatedDelivery": "2023-12-25"
      }
    }
  ]
}
```

Schema Validation

Enforce data integrity for incoming requests using JSON Schema validation.

File: `validation-schema.json`

```json
{
  "name": "auth",
  "endpoints": [
    {
      "method": "POST",
      "path": "/login",
      "validation": {
        "body": {
          "type": "object",
          "required": ["email", "password"],
          "properties": {
            "email": { "type": "string", "format": "email" },
            "password": { "type": "string", "minLength": 8 }
          }
        }
      },
      "response": {
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
      }
    }
  ]
}
```

---

5. Best Practices

1. **Modularize Schemas:** Break down large APIs into smaller, resource-specific schema files (e.g., `users.json`, `products.json`) for better maintainability.
2. **Use Meaningful Status Codes:** Always define appropriate HTTP status codes (200, 201, 400, 404, 500) to accurately simulate API behavior.
3. **Simulate Latency:** Use the `delay` property to test how your frontend handles slow network responses.
4. **Validate Inputs:** Always include validation rules for POST and PUT requests to ensure your mock server rejects invalid data, just like a real backend.
5. **Version Control:** Commit your schema files to Git so your team shares the same mock definitions.

---

6. Troubleshooting

Common Issues

Issue: Server fails to start

- **Cause:** Port 3000 might be in use.
- **Solution:** Use the `--port` flag to specify a different port (e.g., `schemock --port 3001`).

Issue: "Schema validation failed" error

- **Cause:** The JSON in your schema file is malformed.
- **Solution:** Use a JSON validator (like jsonlint.com) to check your schema file syntax.

Issue: Changes not reflecting

- **Cause:** The server might not be watching for file changes.
- **Solution:** Ensure you are running in watch mode (default) or restart the server manually.

Issue: 404 Not Found on defined endpoint

- **Cause:** Mismatch in HTTP method or path.
- **Solution:** Double-check the `method` (GET, POST, etc.) and `path` in your schema definition. Remember paths are case-sensitive.

Generated by Schemock Team