DESIGNING DISTRIBUTED SOFTWARE FOR SCALE

@kigster

by Konstantin Gredeskoul
June 2021

@kigster

FOSSA

WHAT IS SOFTWARE DESIGN?

- What we mean by "software design" is really just a **process of breaking down product requirements** into the necessary software components.
- What is often forgotten is that there are **three ways** to look at any software design:

THREE "VIEWS" INTO SOFTWARE DESIGN

- **Data Model**, also referred to as a "**structure**", and most commonly mapped to rows in the database tables.
- **State Model**, or "state machine" is responsible for the change in structure over time.
- **System Model** is a physical representation of software as it's deployed onto the infrastructure, such as the cloud, or K8S.

DESIGN DECISIONS HAVE LASTING IMPLICATIONS

When you are given a feature to build, and to design a solution, early team collaboration is absolutely essential

- Designing data model in **isolation** often leads to problems such as:
  - Code or Logic Duplication
  - Too rigid or too abstract interfaces
  - Performance problems
  - Lack of foresight, or vice versa: premature optimization

COMMON PROBLEMS WITH SOFTWARE DESIGN

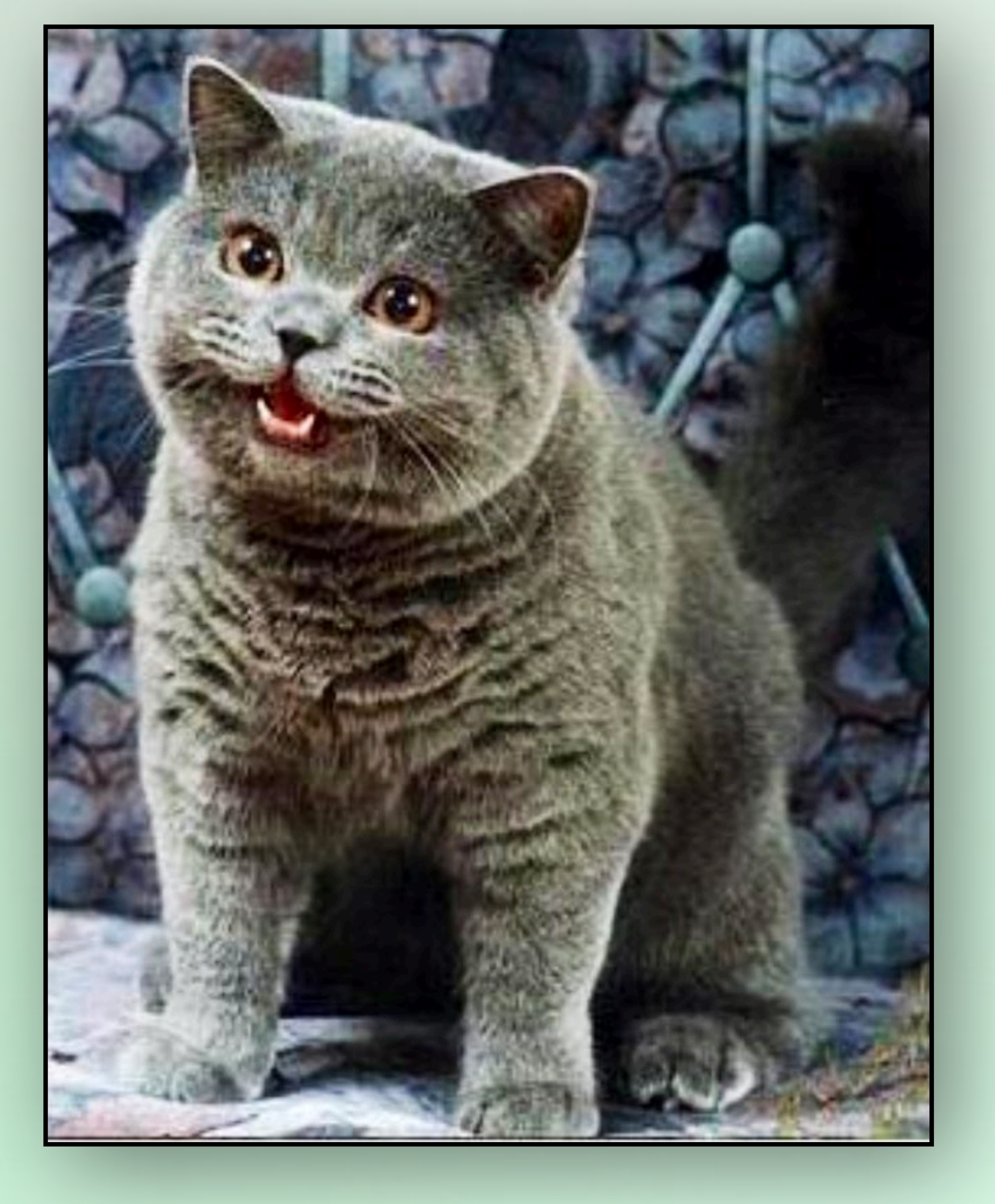
- When we are given a set of product requirements, we begin by applying familiar constructs, such as design patterns, diagrams, database schema relations, etc.
- **Less experienced engineers are more likely to re-apply a smaller set of patterns,** even if the problem calls for a different solution.
- This is completely natural, but it is also one of the reasons that **collaborating with senior engineers on an early design** can lead to choosing a more appropriate solution, while expanding the tool-set of less experienced developers.

WHAT IS AN EFFECTIVE DESIGN PROCESS?

Ultimately, the most important thing is to actually DO the design: meaning: allocate time to think about the solutions, trade-offs, and do that BEFORE the code is written.

- I would argue that **brainstorming in person, with 2-3 engineers next to a white-board** is the ideal setting for doing a sketch of the solution design, i.e the data model (structure) and state.
- While document-based RFCs are a viable alternative, **ideally RFC should be written only AFTER a design brainstorm** takes place, and capture the decisions made there.

DESIGN PATTERNS



SO SOWRY, BUT I ATE THEMS ALL

I CAN HAZ PATTERNS IN GO?

EXPANDING YOUR TOOLKIT • INVALUABLE CLASSICS

Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES
The Addison-Wesley Signature Series
PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE
A MARTIN FOWLER SIGNATURE BOOK
MARTIN FOWLER
WITH CONTRIBUTIONS BY
DAVID RICE,
MATTHEW FOEMMEL,
EDWARD HIEATT,
ROBERT MEE, AND
RANDY STAFFORD
Covers through Version 2.0 OMG UML Standard
UML DISTILLED
THIRD EDITION
A BRIEF GUIDE TO THE STANDARD OBJECT MODELING LANGUAGE
MARTIN FOWLER
Forewords by Cris Kobryn, Grady Booch, Ivar Jacobson, and Jim Rumbaugh
OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
SERIES EDITORS
PRACTICAL OBJECT-ORIENTED DESIGN
AN AGILE PRIMER USING RUBY
SECOND EDITION
SANDI METZ

DESIGN PATTERNS IN GO ■ CREATIONAL PATTERNS

https://github.com/tmrts/go-patterns

Pattern	Description
Builder	Builds a complex object using simple objects
Factory Method	Defers instantiation of an object to a specialized function for creating instances
Object Pool	Instantiates and maintains a group of objects instances of the same type
Singleton	Restricts instantiation of a type to one object

STRUCTURAL PATTERNS

Pattern	Description
Decorator	Adds behavior to an object, statically or dynamically
Proxy	Provides a surrogate for an object to control it's actions

DESIGN PATTERNS IN GO ◘ MESSAGING PATTERNS

https://github.com/tmrts/go-patterns

Pattern	Description
Fan-In	Funnels tasks to a work sink (e.g. server)
Fan-Out	Distributes tasks among workers (e.g. producer)
Publish/Subscribe	Passes information to a collection of recipients who subscribed to a topic

BEHAVIORAL PATTERNS

Pattern	Description
Observer	Provide a callback for notification of events/changes to data
Strategy	Enables an algorithm's behavior to be selected at runtime

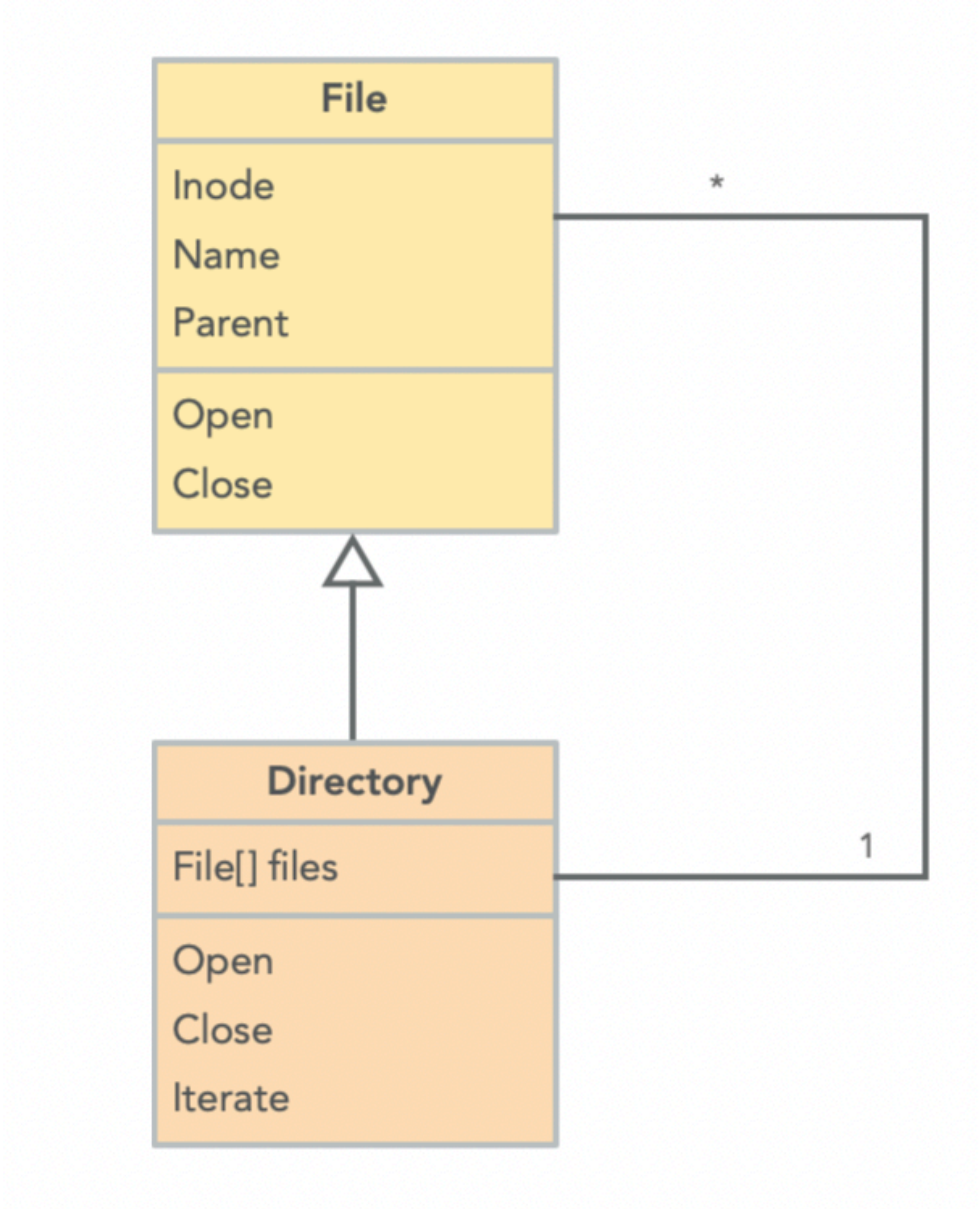
SHARED, HIGHER-ORDER VOCABULARY

- Knowing, and applying design patterns in the right place makes it easy to communicate ideas.
    - If you told me you were working on an **Adapter** or a **Decorator** for some interface, I would instantly have a pretty good idea about the overall design. That's the power of communicating in higher-order constructs.

- The same applies to UML – the graphical representation of the structure, state, interactions, systems architecture, and more.

- Visual documentation is often sufficient to explain how something works, especially over time (which is much harder to explain in words).

AN EXAMPLE: CLASS STRUCTURE

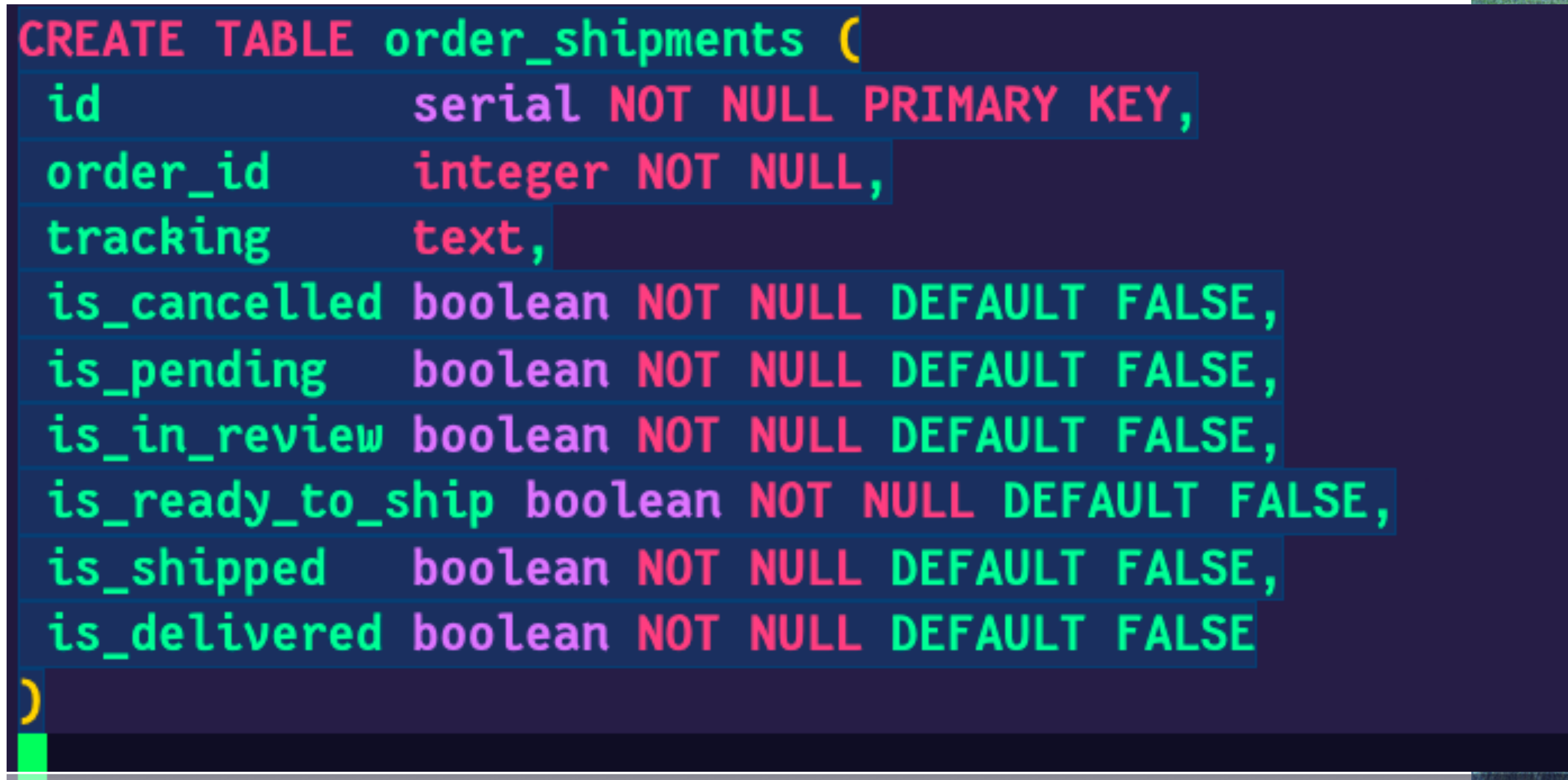


- This shows that **Directory is also a regular File** that has the same properties that File has, but adds additional methods or data.

- If we wanted to store the file system in the database, another Enterprise Design Pattern applies: **Single Table Inheritance.**

AN EXAMPLE: BOOLEAN MADNESS

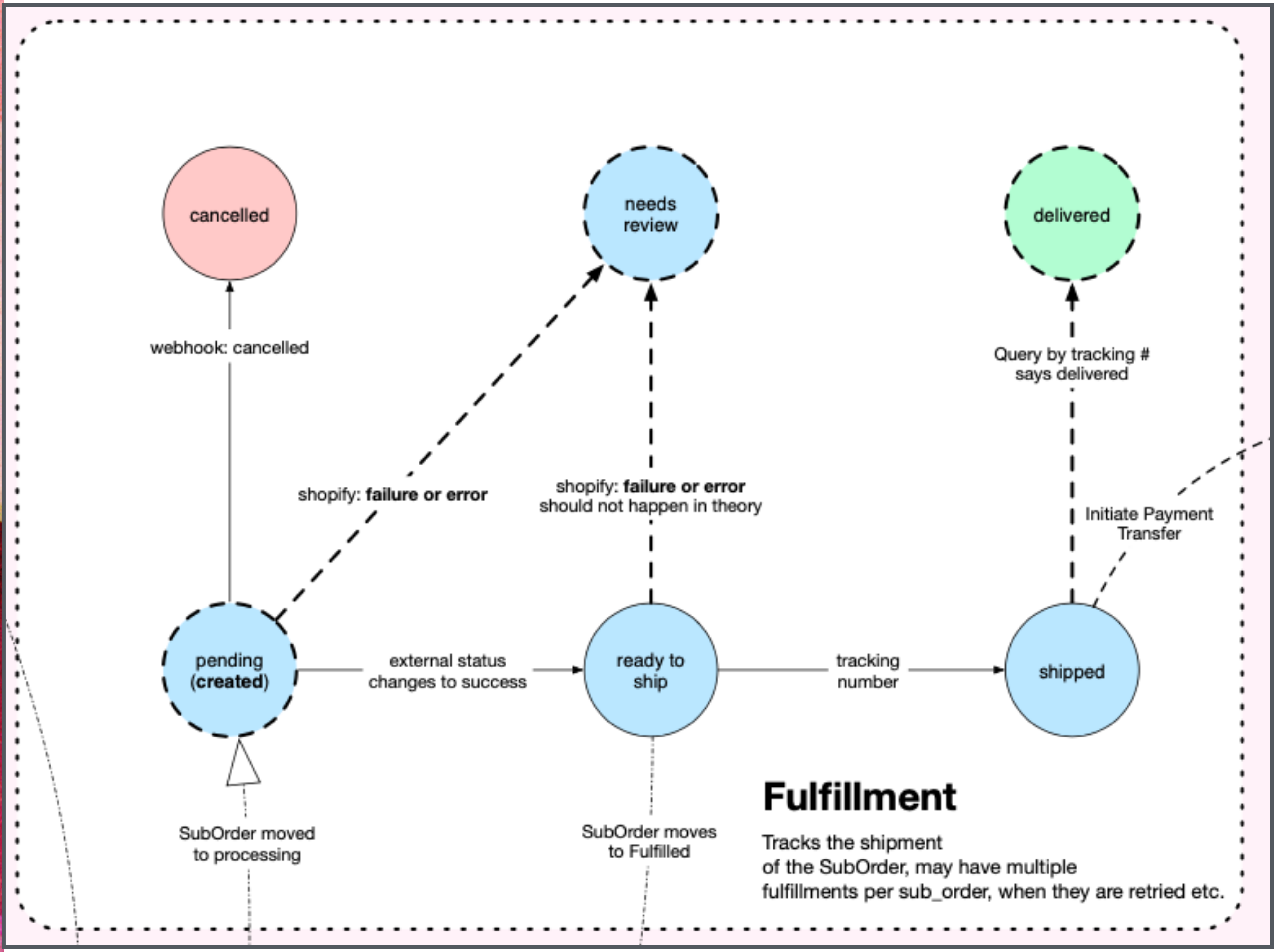
- If we are implementing a model for Order Shipments, this schema comes from an actual project I've worked on.
- For every new state that shipment transitions to, developers added a boolean flag.
- Is that a good solution?
- Let's look at a proper way to do this next...



```
CREATE TABLE order_shipments (
 id           serial NOT NULL PRIMARY KEY,
 order_id     integer NOT NULL,
 tracking     text,
 is_cancelled boolean NOT NULL DEFAULT FALSE,
 is_pending   boolean NOT NULL DEFAULT FALSE,
 is_in_review boolean NOT NULL DEFAULT FALSE,
 is_ready_to_ship boolean NOT NULL DEFAULT FALSE,
 is_shipped   boolean NOT NULL DEFAULT FALSE,
 is_delivered boolean NOT NULL DEFAULT FALSE
)
```

AN EXAMPLE: STATE MACHINE

https://github.com/looplab/fsm


cancelled
needs review
delivered
webhook: cancelled
Query by tracking # says delivered
shopify: failure or error
shopify: failure or error should not happen in theory
Initiate Payment Transfer
pending (created)
external status changes to success
ready to ship
tracking number
shipped
SubOrder moved to processing
SubOrder moves to Fulfilled

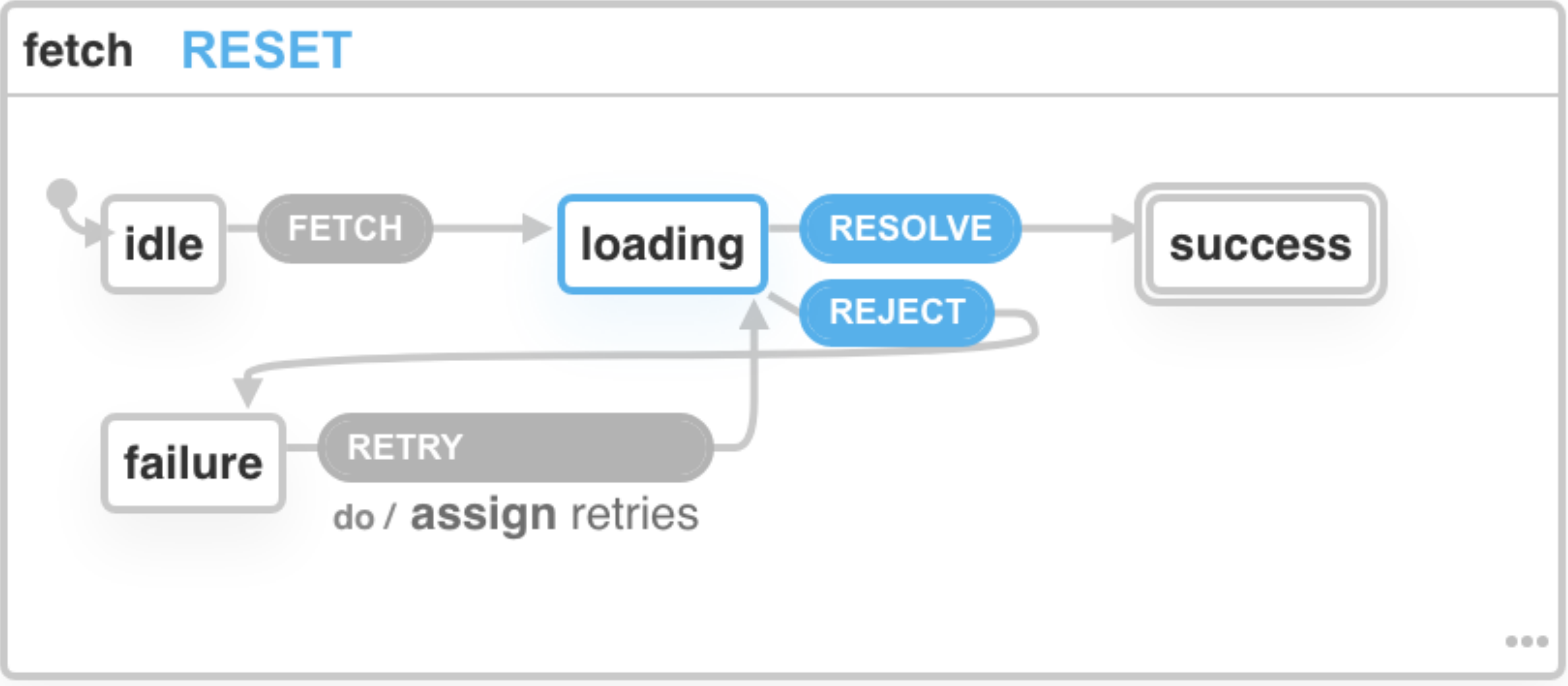
Fulfillment

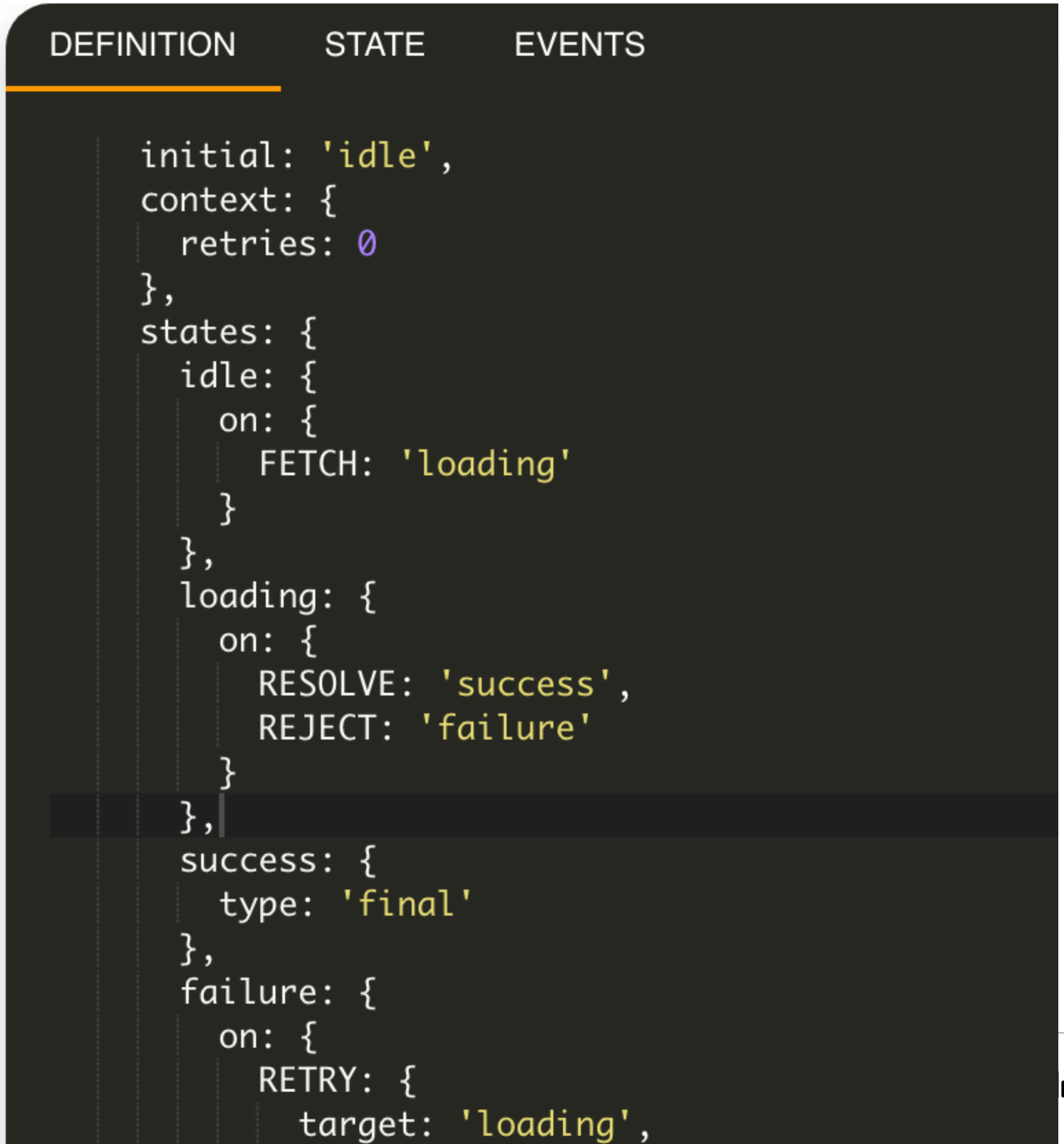
Tracks the shipment of the SubOrder, may have multiple fulfillments per sub_order, when they are retried etc.

- **State Machine** is another incredibly useful design pattern that defines concrete states, together with the valid and invalid transitions between them.
- There could be one or more "starting" states, and one or more "finish" states.
- One of the "finish" states could be "error" or "failed".

AN EXAMPLE: STATE MACHINE IN TYPESCRIPT

https://xstate.js.org/docs/guides/start.html





```
initial: 'idle',
context: {
  retries: 0
},
states: {
  idle: {
    on: {
      FETCH: 'loading'
    }
  },
  loading: {
    on: {
      RESOLVE: 'success',
      REJECT: 'failure'
    }
  },
  success: {
    type: 'final'
  },
  failure: {
    on: {
      RETRY: {
        target: 'loading',
```

EVENTS

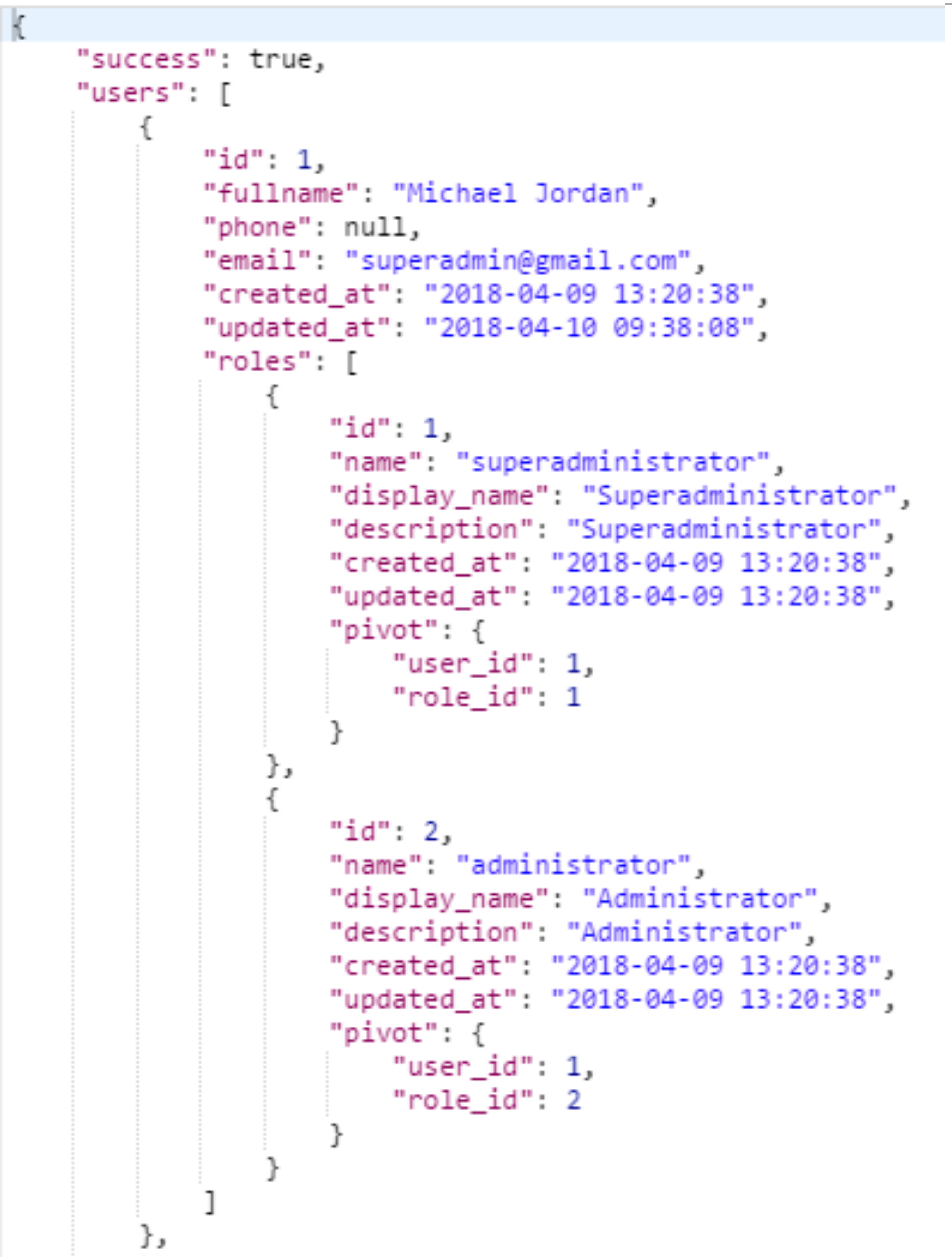
WHAT ARE THEY? WHEN DID THEY HAPPEN? DID I MISS ANYTHING?

EVENTS AS FIRST CLASS CITIZENS

There is a growing trend to define key business events in the application as structs, or hashes, perhaps using JSON with JSON schema validation.

- Whenever you update the database, you are changing state.
- State changing is by definition an important event.
- The event can be represented by a JSON hash that is published to the message bus.

EVENT: EXAMPLE



```
{
    "success": true,
    "users": [
        {
            "id": 1,
            "fullname": "Michael Jordan",
            "phone": null,
            "email": "superadmin@gmail.com",
            "created_at": "2018-04-09 13:20:38",
            "updated_at": "2018-04-10 09:38:08",
            "roles": [
                {
                    "id": 1,
                    "name": "superadministrator",
                    "display_name": "Superadministrator",
                    "description": "Superadministrator",
                    "created_at": "2018-04-09 13:20:38",
                    "updated_at": "2018-04-09 13:20:38",
                    "pivot": {
                        "user_id": 1,
                        "role_id": 1
                    }
                },
                {
                    "id": 2,
                    "name": "administrator",
                    "display_name": "Administrator",
                    "description": "Administrator",
                    "created_at": "2018-04-09 13:20:38",
                    "updated_at": "2018-04-09 13:20:38",
                    "pivot": {
                        "user_id": 1,
                        "role_id": 2
                    }
                }
            ]
        },
```

- If this message is published to eg. RabbitMQ, or Kafka, it's easy to build micro-services that are fully decoupled from user registration.
- In other words, micro-service understands user created event, but the application has no knowledge of the micro-service downstream.

WELL-DESIGNED SOFTWARE IS SIMPLY SOFTWARE THAT IS EASY TO CHANGE.

— DAVE THOMAS

SOFTWARE DESIGN • CONCLUSIONS

- Creating a *lasting design* that can withstand the test of time for any software is hard. It's both science and art and a bit of luck.
- It's rarely a good idea to design *entirely alone in isolation.* Two heads are better than one (that's why we still have Mounted Police on horses).
- **Collaboration** at the design stage has the biggest impact and the return on the investment.
- **UML** and **Design Patterns a**re highly effective tools of collaboration and communication.
- Investing some time into learning how to express the design via UML is **priceless**
  - The book **"**UML Distilled" is only ~ 160 pages long and is one of the most impactful programming books I've ever read.
- Finally, "**Event-driven"** architectures are gaining popularity because they facilitate decoupling of micro-services.

Thanks!

https://github.com/kigster

https://kig.re

twitter.com/kig
github.com/kigster
linkedin.com/in/kigster
slideshare.net/kigster

FOSSA