Data modeled from back- to front-end

Multipurpose, context-dependent and typesafe data model in Scala & Scala.js

October 2017 • Olivier Guerriat

N-SIDE

OPTIMIZING YOUR DECISIONS

- We build tools to solve complex industry challenges and inform decision-making in three key areas:
  - health
  - energy
  - process
- We're growing quickly

Efficiency

- team flexibility
- economies of scale

→

- share code, knowledge, processes...
- Scala everywhere
- good framework

permissions handling

fully reactive UI

charts

high-level

feature-full data model

UI flows

data validation

jobs management

complete control

universal positions

web-based

full of hooks

EDGE

tailored to our needs

built with

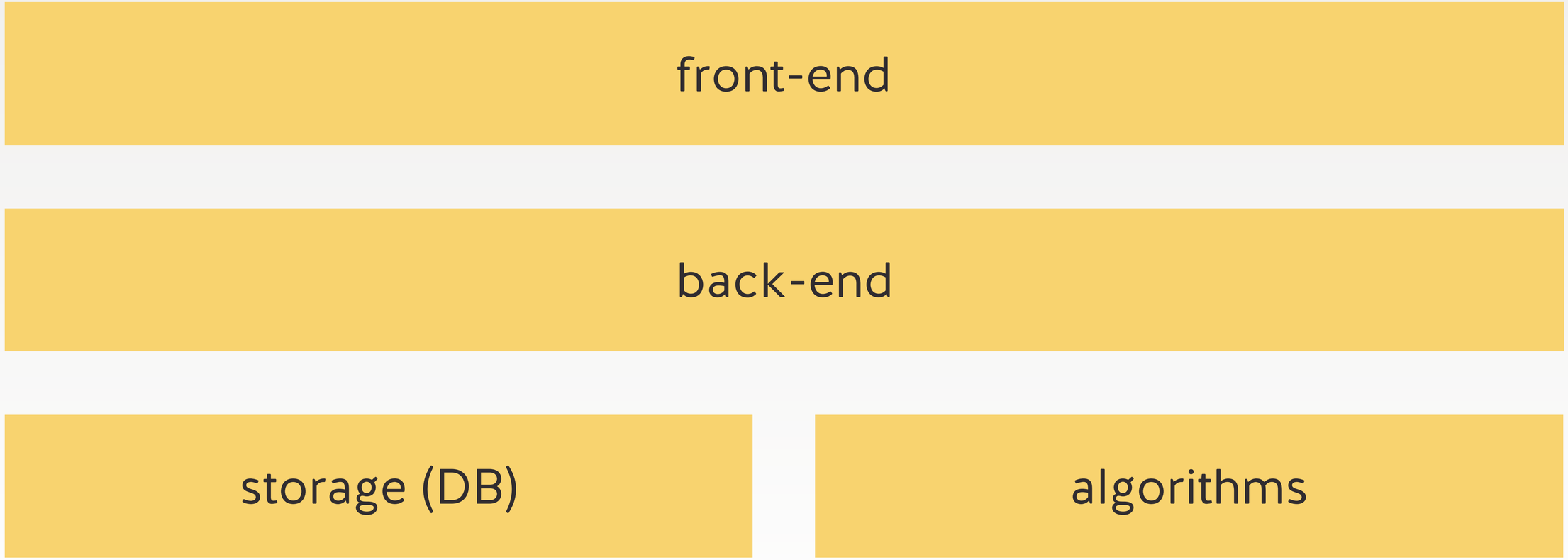
Scala & Scala.js

on top of

Scala & JavaScript

existing libraries

One Single Data Model



Simple Data Structure

Data set

Table

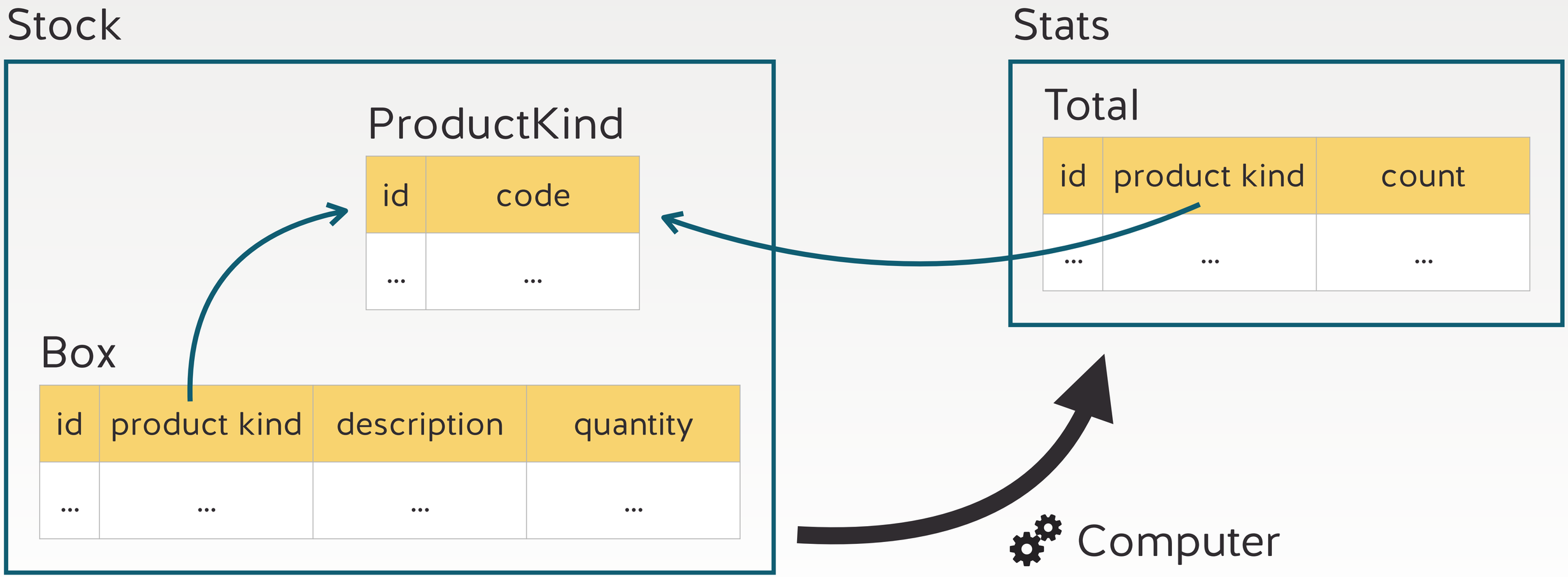
id	attribute 1	attribute 2	attribute 3
...			
...			

id	foo	bar	baz
...			
...			

id	lala	lili	lulu
...	...	...	...
...	...	...	...

- data set ≃ typed xls file
- multiple kinds of data sets
- references between both tables & data sets

Simple App


Stock
ProductKind
id
code
...
...
Box
id
product kind
description
quantity
...
...
...
...
Stats
Total
id
product kind
count
...
...
...
Computer

DEMO

That was nice!

- one single data model
- multiple usages
- multiple DSLs
- lots of automatically generated stuff (diagram, UI, import/export...)

Other Niceties

- automatic JSON/BSON representation
- typed querying
- data migration
  - dedicated DSL
  - checks (are all the changes taken into account?)
- …

Contexts

What's that? How do they work?

Multiple Usages, Multiple DSLs

- mutable <> immutable
- synchronous <> asynchronous <> reactive
- consistent <> potentially inconsistent
- ...

Context

- determines what actions are possible & guarantees about the data
  - when possible, only code the happy path
- one context by kind of usages
  - contexts can inherit from each other
  - "explore the schema" is also an usage

Enablers

1. reuse the schema
2. projected types
3. implicit monkey patching

The Schema is the DSL

- each instance is associated with some data
- avoid macros but still get the nice names (with easy IDE support)

Projected Types

```scala
class ProductKind[C <: Context](
  val id: C#ID,
  val code: C#Attr[String]
) extends DataModelObject[C]
```

≃ type level dot notation

```scala
abstract class Context {
  type Attr[X] <: AbstractAttr[X]
}

class Browsing extends Context {
  type Attr[X] = ImmutableAttr[X]
}

class Editing extends Context {
  type Attr[X] = MutableAttr[X]
}
```

Implicit Monkey Patching

```scala
class Monkey[C <: Context]

object Monkey {
  implicit class EditingMonkeyExt(val m: Monkey[Editing]) extends AnyVal {
    def setAge(nb: Int): Unit = ???
  }
}

val bm: Monkey[Browsing] = ???
bm.setAge(7) // doesn't compile

val em: Monkey[Editing] = ???
em.setAge(42) // does compile
```

Going further

- make the context covariant
- immutable view of the schema
- some typed-checked permissions (e.g. for jobs)

THANK YOU

QUESTIONS?

Olivier Guerriat
olivier@guerriat.be
@olivierg

We're hiring!

N-SIDE
https://n-side.com

Sample project
https://bitbucket.org/nside_projects/contextful-data-model