

An Enrichment System Integration

Scala in a Fintech Domain

Jonathan Deyrson
Scala Functional Designer
@deyrson



Xebia

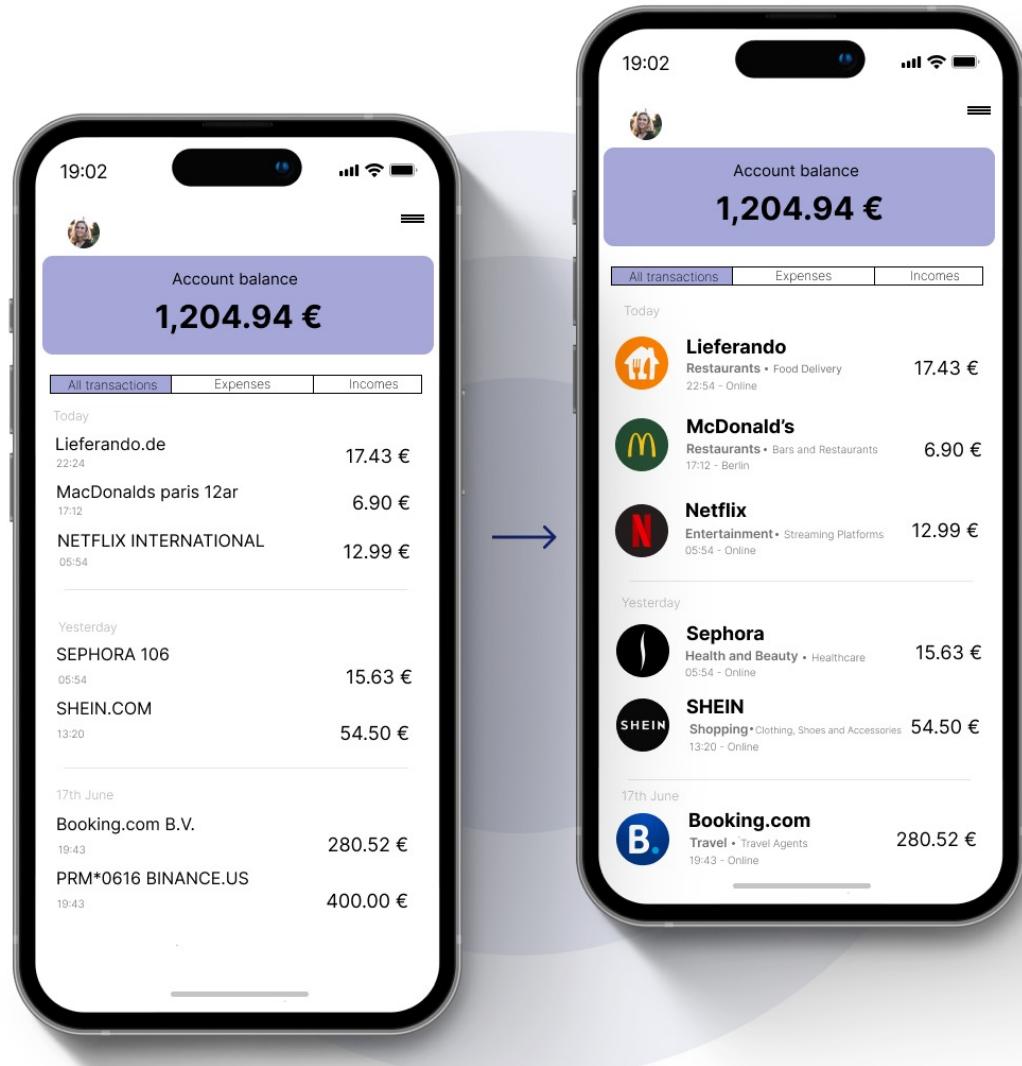


First Part

- Use Case (HLD)
- Context Mapping
- Architectural Design

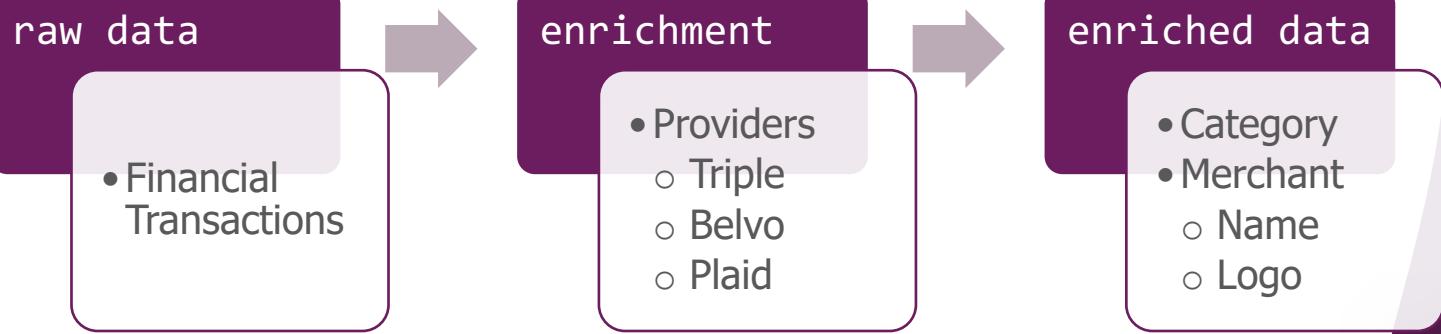
Second Part

- Detailed Design
- Recap
- Scala FP Ecosystem



Data Enrichment as a Product

Raw	sq*Star buc debitcard 20240304 US NY10016	28.99	debit						
Token	sq	*	star	buc	debit	card	20240304	NY10016	US
Predicted	Processor Square	Merchant Starbucks	Transaction type Debit card	Date 20240304	Location NY10016 US				



Enrichment Process

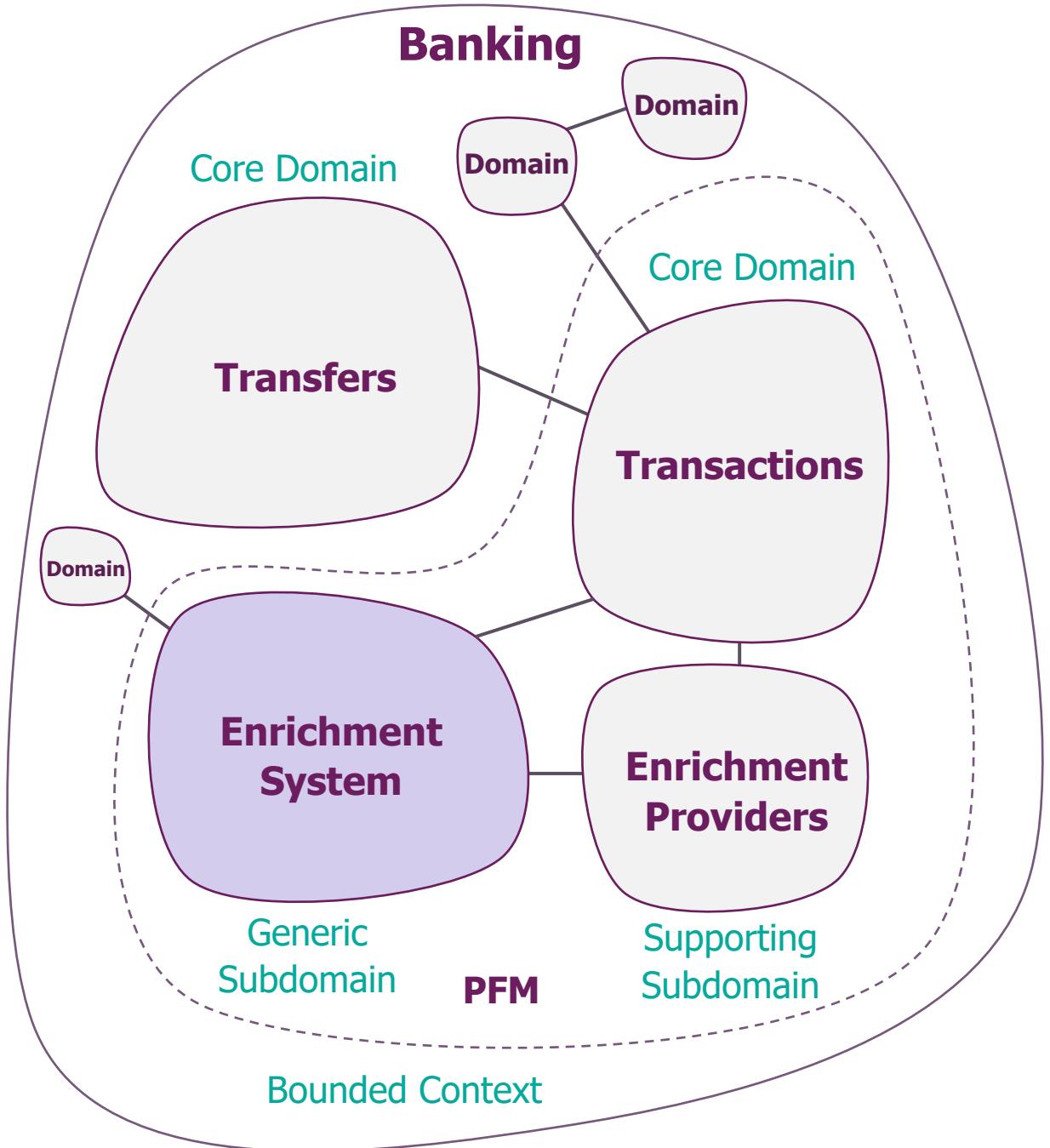
○ ○ ○

```
{
  "raw_transaction": {
    "transaction_id": "e01707b4-e134-562e-8895-eb576883b709",
    "account_id": "5dedaa5a-cb39-406a-a354-d0b8b1219ef8"
  }
}
```

```

○ ○ ○

{
  "enrichment": {
    "categoriser": {
      "categories": {
        "category_name": "eating_out",
        "label": "Eating Out"
      }
    },
    "merchant": {
      "id": "ubereats",
      "merchant_name": "Uber Eats",
      "merchant_logo":
        "https://assets.enrichment.com/provider-datasci-
        images/merchant_logos/c16f192b-72de-426e-acd1-
        6ba680b7882b/v1/ubereats.jpeg"
    }
  }
}
  
```



Domain and Scope

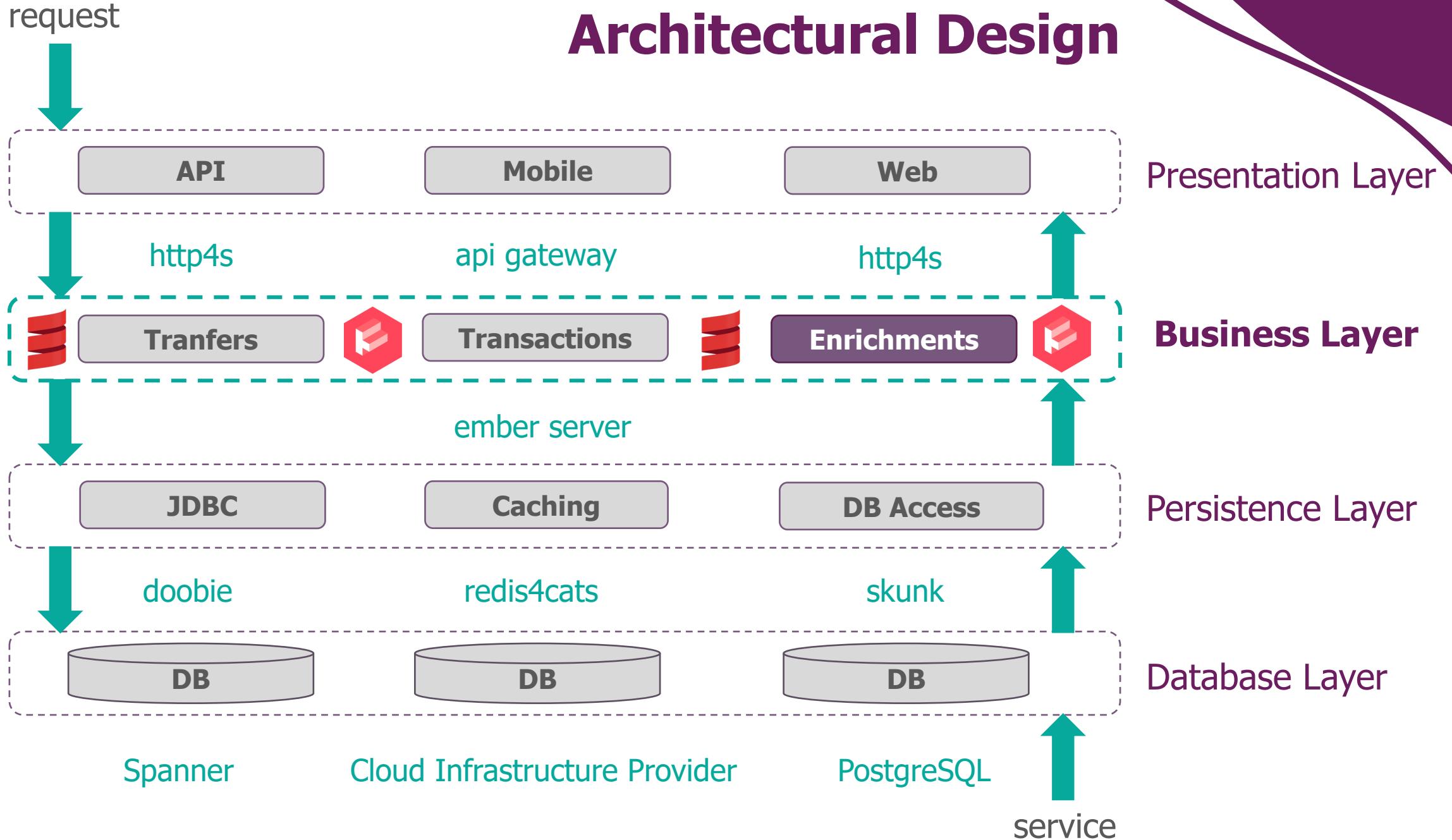
Domain-Driven Design (DDD)

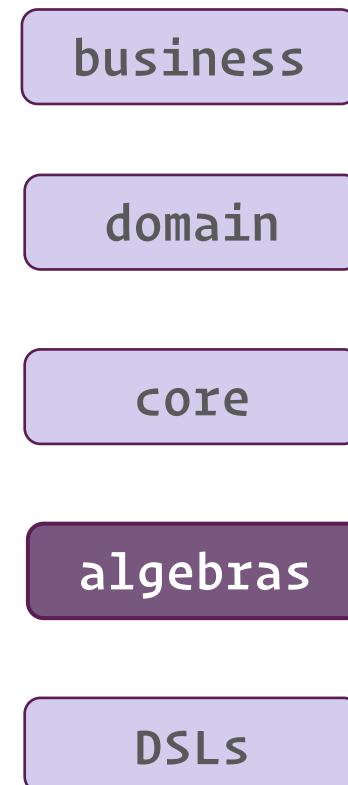
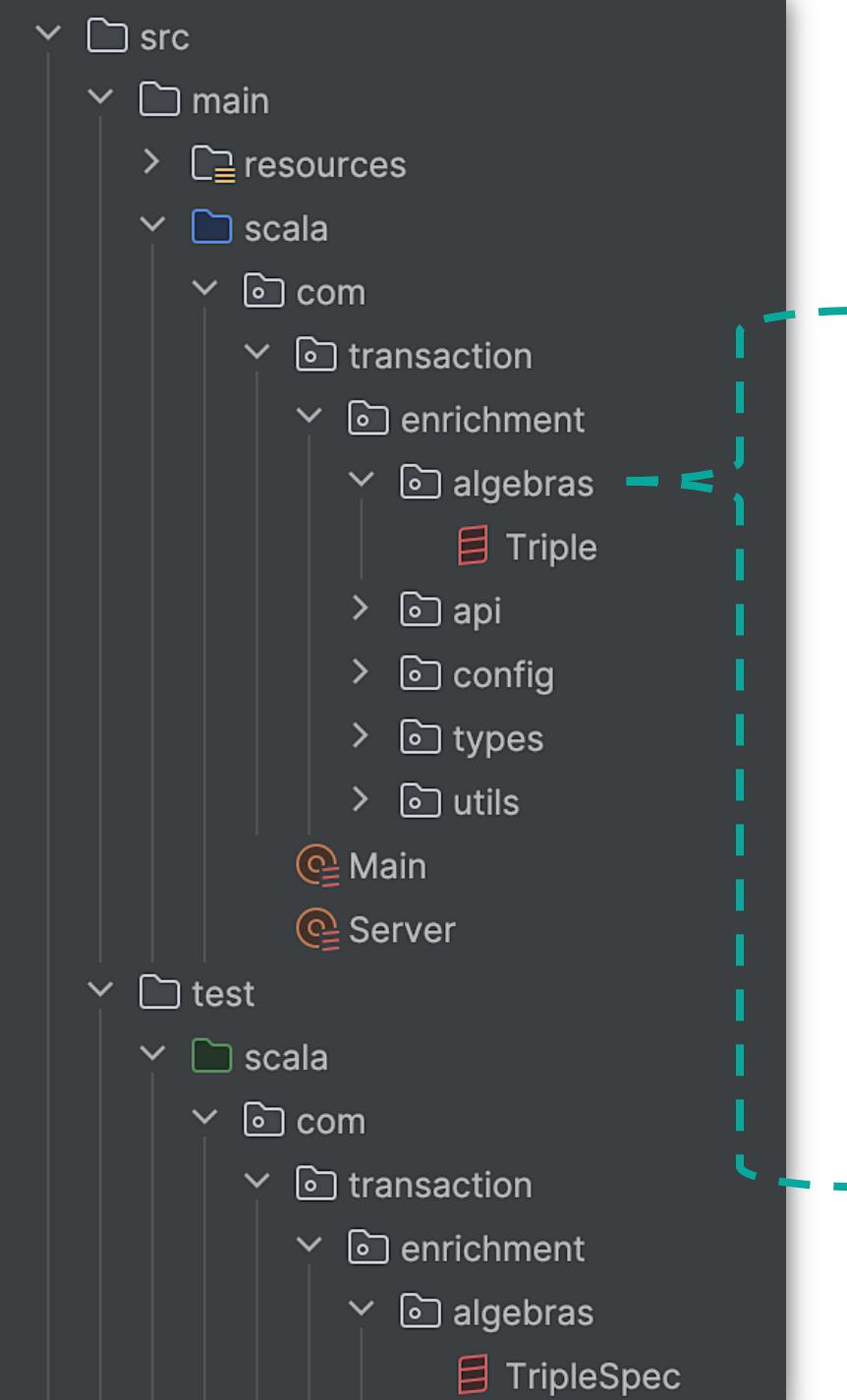
- Strategic Patterns (Architectural Design)
- Tactical Patterns (Detailed Design)

Critical Software Systems

- Safety Critical
- Mission Critical
- **Business Critical**

Architectural Design





Project Scaffolding

A minimal example structure

http4s giter8 template

This template is fixed on `cats.effect.IO`. For a final tagless version, see [http4s.g8](#).

Instructions

Generate an http4s service on the ember backend with Circe.

1. [Install sbt](#)
2. Create your project:
 - Scala 2: `sbt new http4s/http4s-io.g8`
 - Scala 3: `sbt new http4s/http4s-io.g8 --branch 0.23-scala3`
 - Java 8: `sbt new http4s/http4s-io.g8 --branch 0.23-java8`
3. `cd quickstart`
4. `sbt run`
5. `curl http://localhost:8080/hello/$USER`
6. [Learn more](#)

Design Styles

Term-Level
Programming

values

First-Class Citizens

functions

types

Type-Level
Programming

B. Liskov, 1974

Concrete Data
Types

Abstract Data
Types (ADTs)

Algebraic Data
Types (ADTs)

Type Classes

Type of a
Type

Primitive
Data Types

Built-in
Data Types

Class

User-
Defined
Types

Sum
Types

Product
Types

Patterns

Effect
Systems

Kinds

Higher-
Kinded
Types (HKT)

Dependent
Types

- Type checking
- Type-safe

- Expressive type system
- Rich type system

Static Type
System

Object-Oriented
Design

Algebraic
Design *

Type-Level
Design

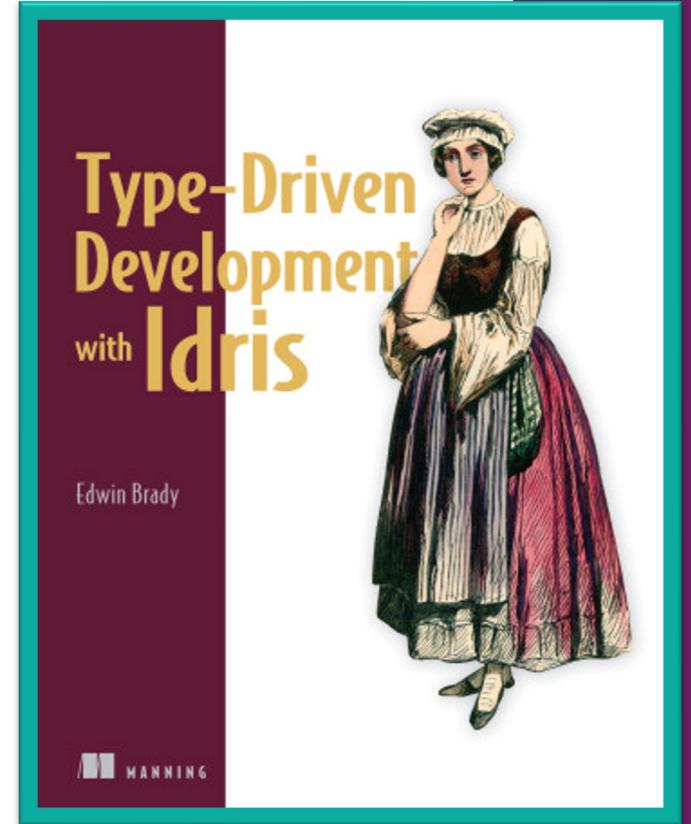
Static Type
System

The Expression Problem

* Just as *algebra* involves operations that follow well-defined rules (like addition and multiplication), *algebraic design* structures data and computations according to similar rules, making them *mathematically sound*.

“Types allow you to make models explicit in code and ensure that your implementation of a program matches the model in your head.”

—Edwin Brady, 2017



Code as Design: Three Essays by Jack W. Reeves

[PDF of All 3 Essays](#) | [Discussion Page](#)

The following essays by [Jack W. Reeves](#) offer three perspectives on a single theme, namely that programming is fundamentally a design activity and that the only final and true representation of "the design" is the source code itself. This simple assertion gives rise to a rich discussion —one which Reeves explores fully in these three essays.

https://www.developerdotstar.com/mag/articles/reeves_design_main.html

Software Design

- Architectural Design
- Detailed Design

Big design up front (BDUF) is a [software development](#) approach in which the program's design is to be completed and perfected before that program's implementation is started. It is often associated with the [waterfall model](#) of software development.

https://en.wikipedia.org/wiki/Big_design_up_front

The lost art of Detailed Design Caveat

Minimal Use Case

An Enrichment System Integration



*This use case explores an **approximation** of enriching raw transaction data by **integrating** with a transaction **enrichment provider** to obtain detailed categories.*



*The core idea is to take **generic financial transactions**, each containing data such as a **transaction ID** and **account ID** and enrich them.*



*Enriched transactions must include categories such as "**Food**," "**Entertainment**," "**Groceries**," "**Health**," "**Travel**," as well as the **merchant's name** and an optional **merchant logo**.*



*Enriched transactions will significantly **enhance** the usability of the transaction data for **end-users** by adding recognizable and transaction details.*

Scenario 1

```
○ ○ ○  
1 import cats.effect.IO  
2 import io.chrisdavenport.fuuid.FUUID  
3 import org.http4s.{Method, Request, Uri}  
4 import org.http4s.client.Client  
5  
6 // --- ADTs (Product Types) ---  
7 case class Transaction(id: String, accountId: FUUID)  
8  
9 trait EnrichmentService {  
10   def submitTransaction(transaction: Transaction): IO[Transaction]  
11 }  
12  
13 object EnrichmentService {  
14  
15   def apply(baseUri: Uri, httpClient: Client[IO]): EnrichmentService = new EnrichmentService {  
16  
17     def submitTransaction(transaction: Transaction): IO[Transaction] = {  
18  
19       val request = Request[IO](Method.POST, baseUri / "enrich")  
20         .withEntity(transaction)  
21  
22       httpClient.expect[Transaction](request)  
23     }  
24   }  
25 }  
26
```

○ ○ ○

```
1 import cats.effect.IO
2 import cats.implicits._
3 import io.chrisdavenport.fuuid.FUUID
4 import org.http4s.{Method, Request, Uri}
5 import org.http4s.client.Client
6
7 // --- ADTs (Product Types) ---
8
9 case class Transaction(id: String, accountId: FUUID)
10
11 case class TripleEnrichedTransaction(
12   id: String,
13   accountId: FUUID,
14   category: String,
15   merchantName: Option[String],
16   merchantLogo: Option[String],
17   customCategory: Option[String]
18 )
19
20 // --- ADTs (Sum Types) ---
21
22 sealed trait EnrichmentStatus
23 case object Success extends EnrichmentStatus
24 case object Failure extends EnrichmentStatus
25
```

Scenario 2



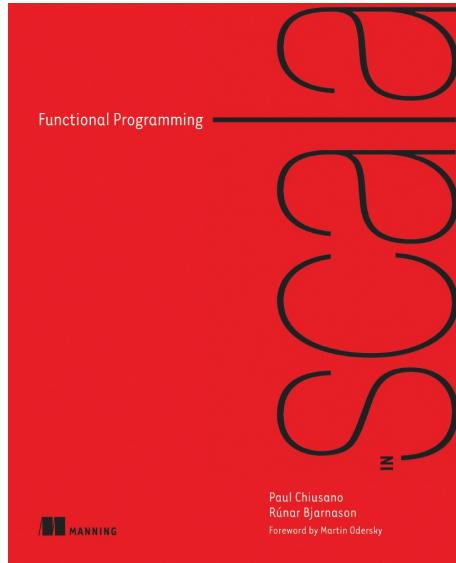
```
26 // --- I0-based Implementation for Triple Enrichment ---
27
28 trait TripleEnrichmentService {
29   def enrich(transactions: List[Transaction]): I0[(List[TripleEnrichedTransaction], EnrichmentStatus)]
30 }
31
32 object TripleEnrichmentService {
33
34   def apply(baseUri: Uri, httpClient: Client[I0]): TripleEnrichmentService = new TripleEnrichmentService {
35
36     def enrich(transactions: List[Transaction]): I0[(List[TripleEnrichedTransaction], EnrichmentStatus)] = {
37
38       val request = Request[I0](Method.POST, baseUri / "enrich" / "transactions")
39         .withEntity(transactions)
40
41       httpClient.expect[List[TripleEnrichedTransaction]](request)
42         .map(enrichedTransactions => {
43           val enrichedWithCustomCategory = enrichedTransactions.map { t =>
44             t.copy(customCategory = Some("Custom Category Example"))
45           }
46           (enrichedWithCustomCategory, Success)
47         })
48         .handleError(_ => (List.empty[TripleEnrichedTransaction], Failure))
49     }
50   }
51 }
```

Scenario 3

```
1 import cats.effect.IO
2 import cats.implicits._
3 import io.chrisdavenport.fuuid.FUUID
4
5 case class Transaction(id: String, accountId: FUUID)
6
7 case class TripleEnrichedTransaction(
8   id: String,
9   accountId: FUUID,
10  category: String,
11  merchantName: Option[String],
12  merchantLogo: Option[String],
13  customCategory: Option[String]
14 )
15
16 case class BelvoEnrichedTransaction(
17   id: String,
18   accountId: FUUID,
19   merchantReputationStars: Option[Int]
20 )
21
22 case class FullyEnrichedTransaction(
23   id: String,
24   accountId: FUUID,
25   category: String,
26   merchantName: Option[String],
27   merchantLogo: Option[String],
28   customCategory: Option[String],
29   merchantReputationStars: Option[Int]
30 )
31
```



```
32 trait EnrichmentService[T] {  
33   def enrich(transactions: List[Transaction]): IO[List[T]]  
34 }  
35  
36 class ComposedEnrichmentService(  
37   tripleService: EnrichmentService[TripleEnrichedTransaction],  
38   belvoService: EnrichmentService[BelvoEnrichedTransaction]  
39 ) {  
40  
41   def enrich(transactions: List[Transaction]): IO[List[FullyEnrichedTransaction]] = {  
42     val tripleIO = tripleService.enrich(transactions)  
43     val belvoIO = belvoService.enrich(transactions)  
44  
45     (tripleIO, belvoIO).parMapN { (tripleTxs, belvoTxs) =>  
46       for {  
47         tripleTx <- tripleTxs  
48         belvoTx <- belvoTxs.find(_.id == tripleTx.id)  
49       } yield FullyEnrichedTransaction(  
50         tripleTx.id,  
51         tripleTx.accountId,  
52         tripleTx.category,  
53         tripleTx.merchantName,  
54         tripleTx.merchantLogo,  
55         tripleTx.customCategory,  
56         belvoTx.merchantReputationStars  
57       )  
58     }.handleError(_ => List.empty[FullyEnrichedTransaction])  
59   }  
60 }  
61
```



“Functional Programming is a restriction on how we write programs, but not on what programs we can express.”

—P. Chiusano & R. Bjarnason

Recap

- What makes Scala a good addition to your skill set and tech stack?

Java 8

- Lambda Expressions

Java 9

- Modules

Java 17

- Sealed Classes

*“An ADT relates a structural idea
to an abstraction, and abstraction
is the way we design programs.”*

—Barbara Liskov



How Data Abstraction changed Computing forever
Barbara Liskov, 2019 | TEDxMIT

https://www.youtube.com/watch?v=_jTc1BTFdIo

*“This paper presents an approach which allows the set of **built-in abstractions** to be **augmented** when the need for a new **data abstraction** is discovered.”*

—B. Liskov, 1974

PROGRAMMING WITH ABSTRACT DATA TYPES

Barbara Liskov
Massachusetts Institute of Technology
Project MAC
Cambridge, Massachusetts

Stephen Zilles
Cambridge Systems Group
IBM Systems Development Division
Cambridge, Massachusetts

*“Programming starts with **types** and **functions**. ”*

—Bartosz Milewski

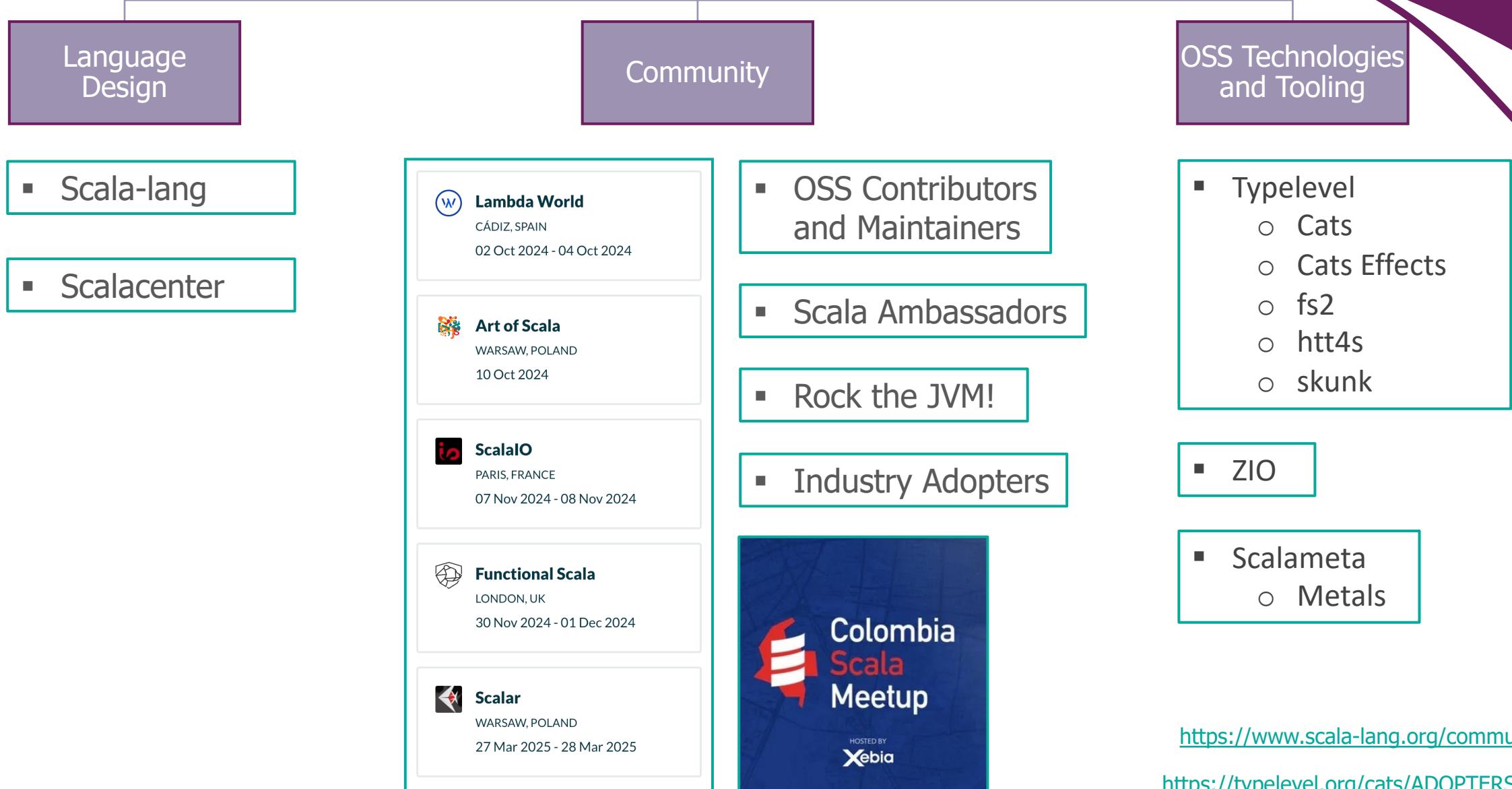
The Dao of Functional Programming

Bartosz Milewski

(Last updated: September 9, 2024)

<https://github.com/BartoszMilewski/Publications/blob/master/TheDaoOfFP.DaoFP.pdf>

Scala FP Ecosystem





Thanks!