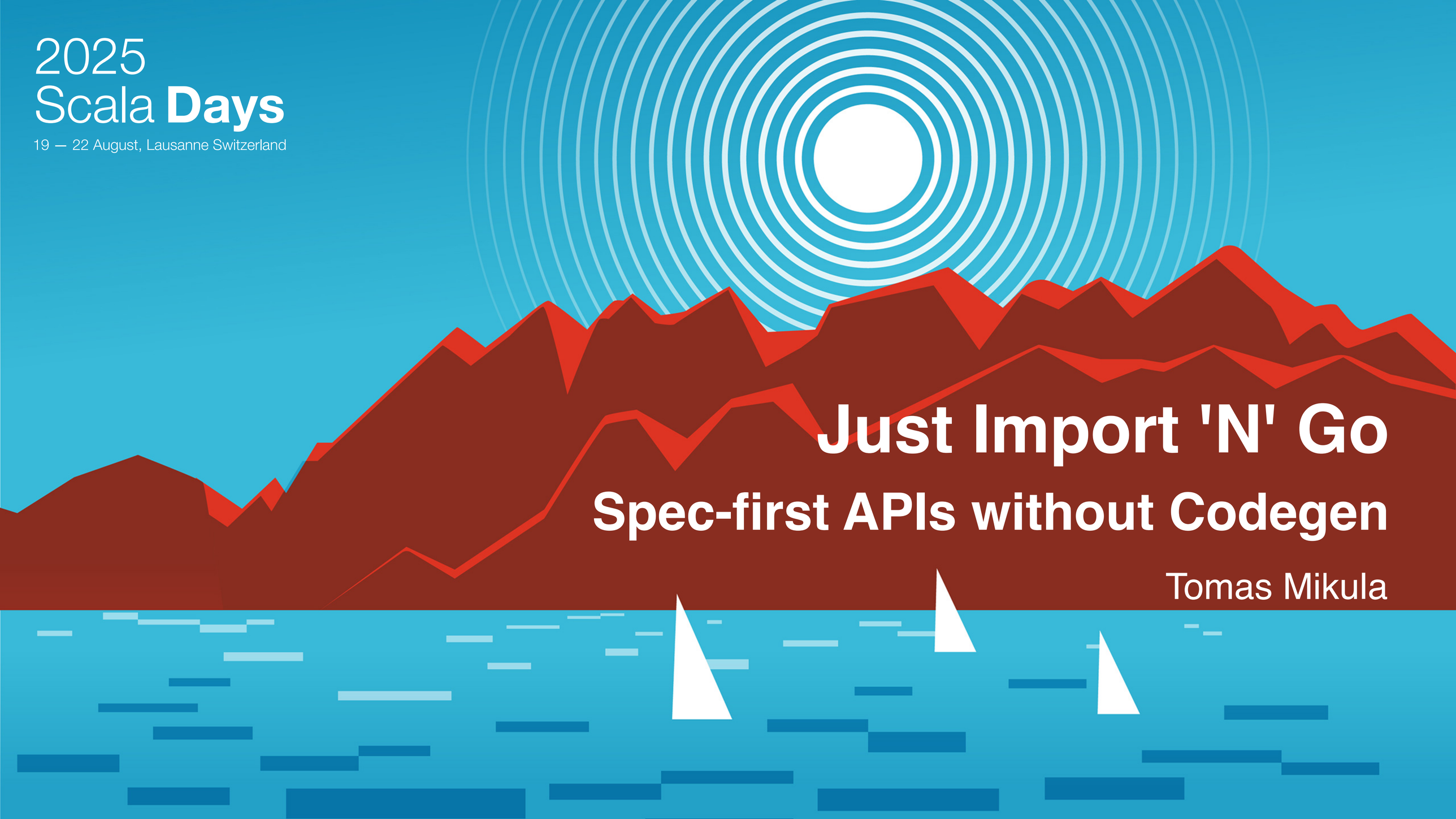

2025
Scala Days
19 — 22 August, Lausanne Switzerland
Just Import 'N' Go
Spec-first APIs without Codegen
Tomas Mikula

Just Import ’N’ Go (JING)

a *library* for programming against OpenAPI specification

Just Import ’N’ Go (JING)

a *library* for programming against OpenAPI specification

Most libraries are not worth *learning!*

Just Import ’N’ Go (JING)

a *library* for programming against OpenAPI specification

Most libraries are not worth *learning!*

including JING

Just Import ’N’ Go (JING)

a *library* for programming against OpenAPI specification

Most libraries are not worth *learning!*

including JING

I am not going to *teach* you JING.

Most libraries are not worth *learning!*

Most tools are not worth *learning!*

Most tools are not worth *learning!*

Most tools are not worth *learning!*

Most tools are not worth *learning!*

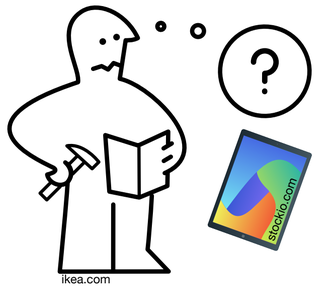
Imagine your favorite app needing an *instruction manual*

Most tools are not worth *learning!*

Imagine your favorite app needing an *instruction manual*

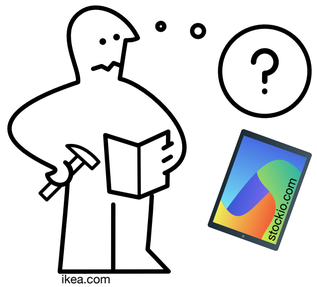
Most tools are not worth *learning!*

Imagine your favorite app needing an *instruction manual*



Most tools are not worth *learning!*

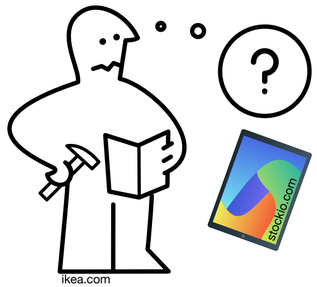
Imagine your favorite app needing an *instruction manual*



before

Most tools are not worth *learning!*

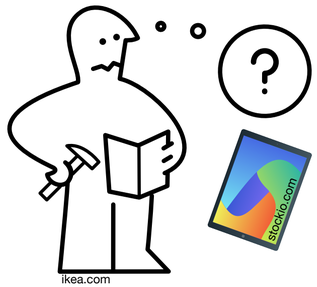
Imagine your favorite app needing an *instruction manual*



before playing a song

Most tools are not worth *learning!*

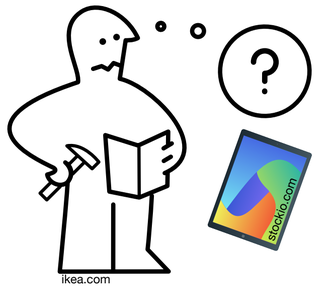
Imagine your favorite app needing an *instruction manual*



before ordering food

Most tools are not worth *learning!*

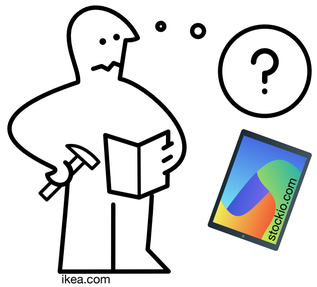
Imagine your favorite app needing an *instruction manual*



before booking a ride

Most tools are not worth *learning!*

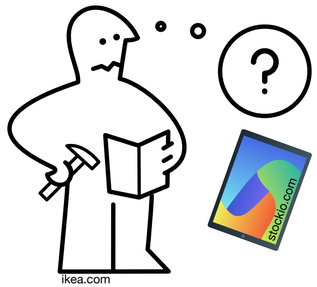
Imagine your favorite app needing an *instruction manual*



before making a payment

Most tools are not worth *learning!*

Imagine your favorite app needing an *instruction manual*

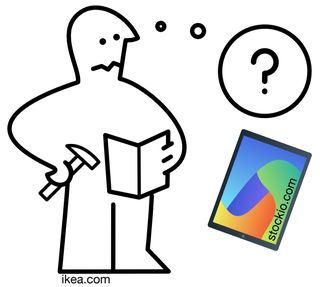


before making a payment

They should just get the job done!

Most tools are not worth *learning!*

Imagine your favorite app needing an *instruction manual*



before making a payment

They should just get the job done!

without the need to be *learnt*

Most tools are not worth *learning!*

Most remaining should not need *upfront* investment

Most remaining should not need *upfront* investment

Learn as you go!

Just Import ’N’ Go (JING)

a *library* for programming against OpenAPI specification

Just Import ’N’ Go (JING)

a set of *principles* for library design

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability

Just Import ’N’ Go (JING)

a set of *principles* for library design

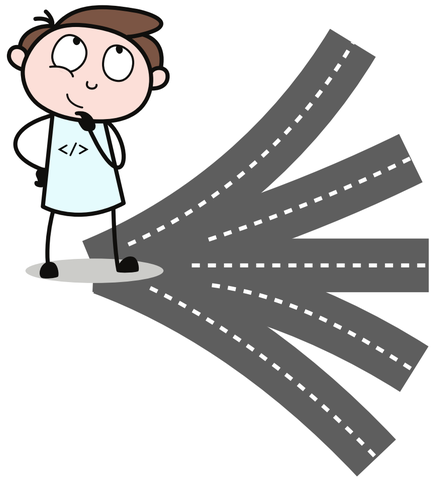
- Discoverability — Via **local** exploration from where I am now

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability

Via **local** exploration from where I am now

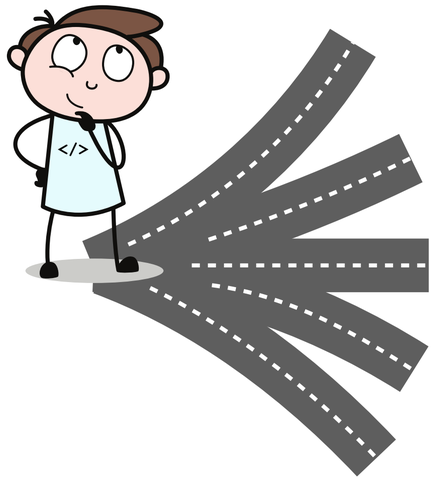


Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability

Via **local** exploration from where I am now



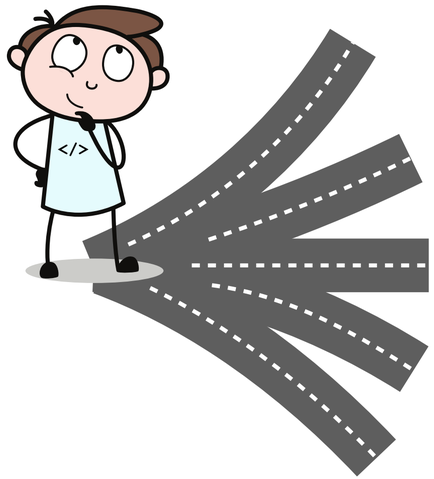
Type the dot

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability

Via **local** exploration from where I am now



Type the dot

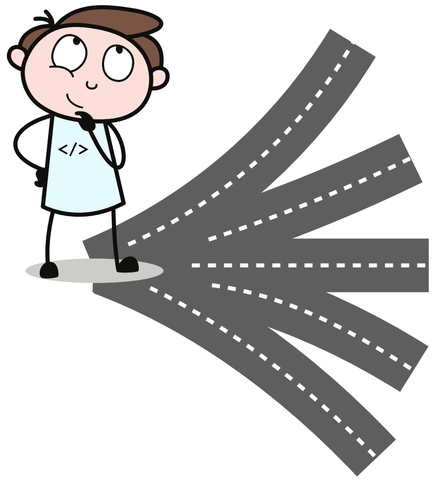
Look in the companion object

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability

Via **local** exploration from where I am now



Type the dot

Look in the companion object

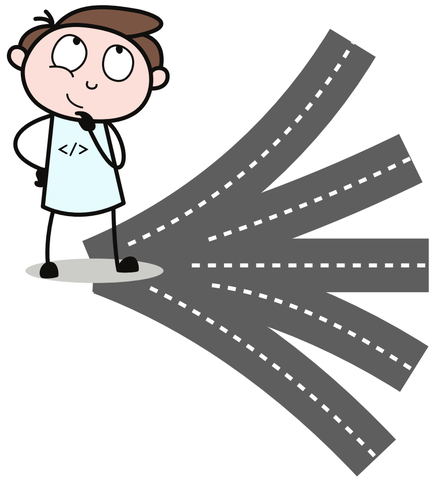
Pattern match

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability

Via **local** exploration from where I am now



Type the dot

Look in the companion object

Pattern match

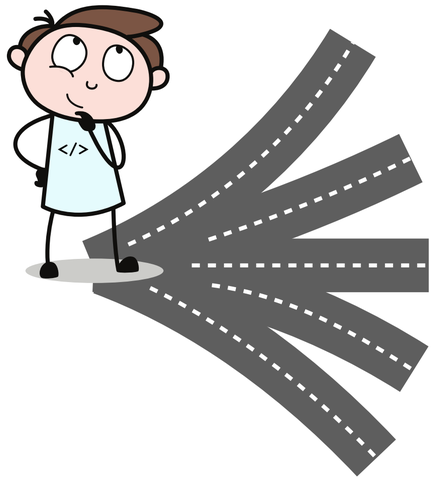
Peek at Scaladoc

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability

Via **local** exploration from where I am now



Type the dot

Look in the companion object

Pattern match

Peek at Scaladoc

Compiler error message

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability — Via **local** exploration from where I am now

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability
- Guidance

Via **local** exploration from where I am now

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability — Via **local** exploration from where I am now
- Guidance — Facilitate introduction to the domain

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability — Via **local** exploration from where I am now
- Guidance — Facilitate introduction to the domain
- Ramp, but no ceiling

Just Import ’N’ Go (JING)

a set of *principles* for library design

• Discoverability	Via **local** exploration from where I am now
• Guidance	Facilitate introduction to the domain
• Ramp, but no ceiling	Simple things simple, complex things possible

Just Import 'N' Go (JING)

a set of *principles* for library design

- Discoverability — Via **local** exploration from where I am now
- Guidance — Facilitate introduction to the domain
- Ramp, but no ceiling — Simple things simple, complex things possible
- Safety

Just Import ’N’ Go (JING)

a set of *principles* for library design

• Discoverability	Via **local** exploration from where I am now
• Guidance	Facilitate introduction to the domain
• Ramp, but no ceiling	Simple things simple, complex things possible
• Safety	Provide reasonable guardrails

Just Import ’N’ Go (JING)

a set of *principles* for library design

• Discoverability	Via **local** exploration from where I am now
• Guidance	Facilitate introduction to the domain
• Ramp, but no ceiling	Simple things simple, complex things possible
• Safety	Provide reasonable guardrails
• Hide unnecessary details	

Just Import ’N’ Go (JING)

a set of *principles* for library design

• Discoverability	Via **local** exploration from where I am now
• Guidance	Facilitate introduction to the domain
• Ramp, but no ceiling	Simple things simple, complex things possible
• Safety	Provide reasonable guardrails
• Hide unnecessary details	Operate at the right level of abstraction

Just Import ’N’ Go (JING)

a set of *principles* for library design

• Discoverability	Via **local** exploration from where I am now
• Guidance	Facilitate introduction to the domain
• Ramp, but no ceiling	Simple things simple, complex things possible
• Safety	Provide reasonable guardrails
• Hide unnecessary details	Operate at the right level of abstraction
• Zero setup	

Just Import ’N’ Go (JING)

a set of *principles* for library design

• Discoverability	Via **local** exploration from where I am now
• Guidance	Facilitate introduction to the domain
• Ramp, but no ceiling	Simple things simple, complex things possible
• Safety	Provide reasonable guardrails
• Hide unnecessary details	Operate at the right level of abstraction
• Zero setup	Just Import ’N’ Go!

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

Just Import ’N’ Go (JING)

a set of *principles* for library design

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

Just Import 'N' Go (JING)

a set of *principles* for library design

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

Good News!

Scala ecosystem already decent at JING principles

Just Import 'N' Go (JING)

a set of *principles* for library design

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

Good News!

Scala ecosystem already decent at JING principles

In an ideal world

Just Import 'N' Go (JING)

a set of *principles* for library design

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

Good News!

Scala ecosystem already decent at JING principles

In an ideal world

Knowing Scala is all you need.
Pick up the rest through JING.

Just Import ’N’ Go (JING)

a *library* for programming against OpenAPI specification

Just Import 'N' Go (JING)

a *library* for programming against OpenAPI specification

OpenAPI

Just Import 'N' Go (JING)

a *library* for programming against OpenAPI specification

OpenAPI

a specification language for HTTP APIs

Just Import 'N' Go (JING)

a *library* for programming against OpenAPI specification

OpenAPI

a specification language for HTTP APIs

Let JING guide us through the rest!

Just Import 'N' Go (JING)

a *library* for programming against OpenAPI specification

OpenAPI

a specification language for HTTP APIs

Let JING guide us through the rest!

Just Import 'N' Go (JING)

a *library* for programming against OpenAPI specification

i.e. specification first

OpenAPI

a specification language for HTTP APIs

Let JING guide us through the rest!

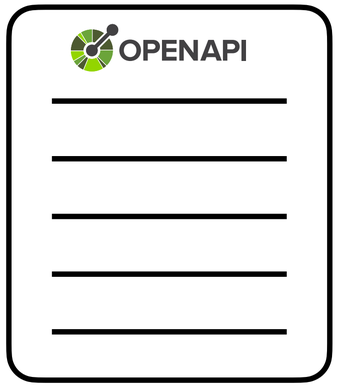
Spec-first APIs

Spec-first APIs

SotA:

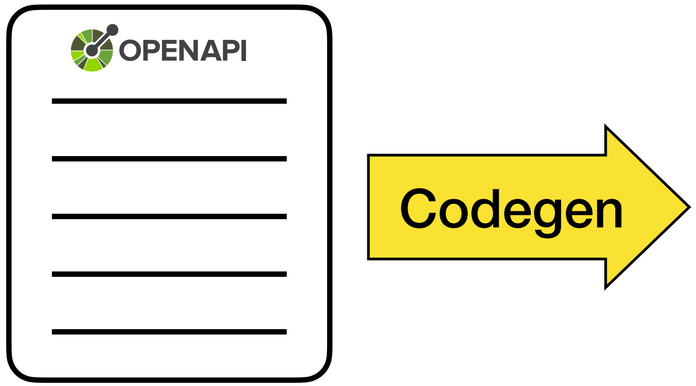
Spec-first APIs

SotA:



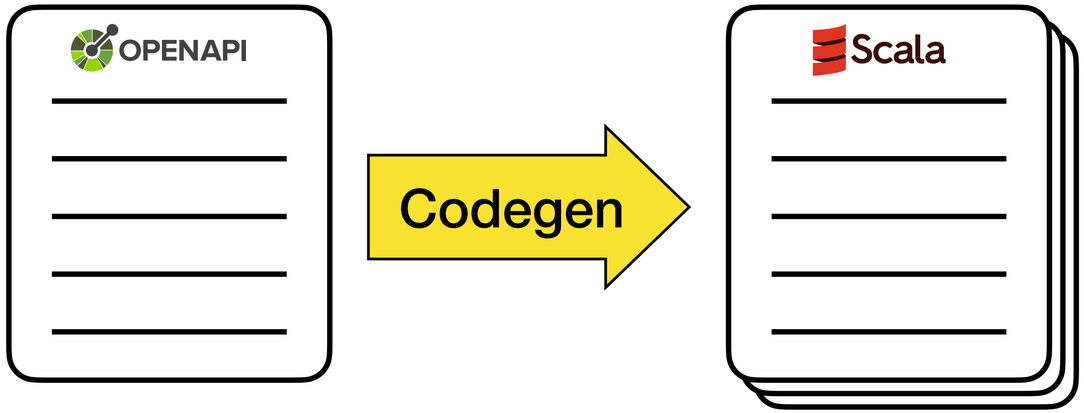
Spec-first APIs

SotA:



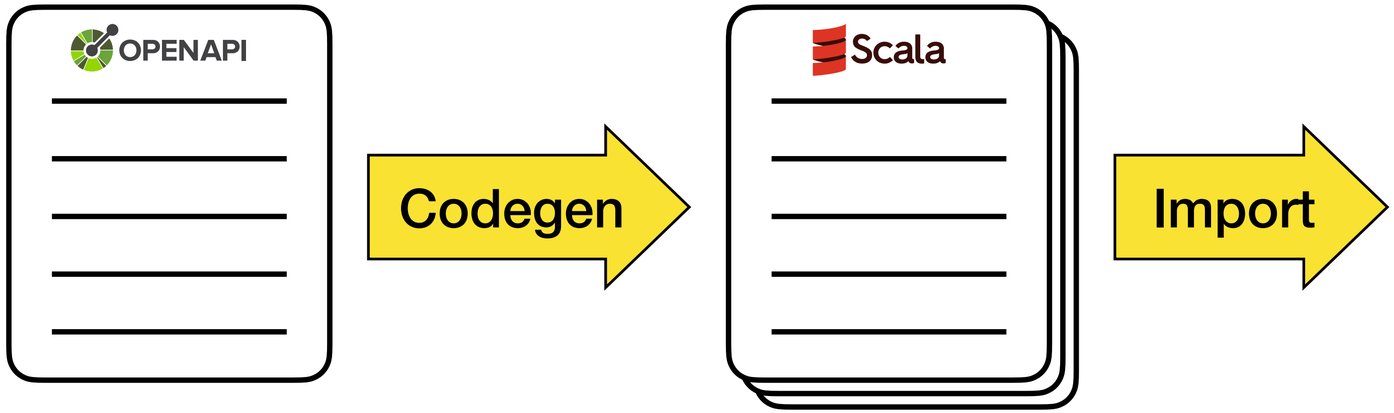
Spec-first APIs

SotA:



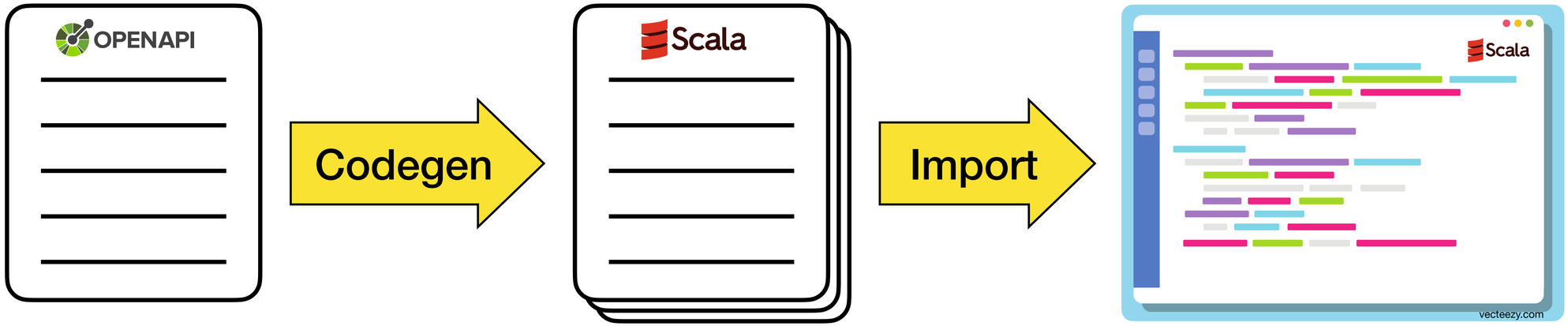
Spec-first APIs

SotA:



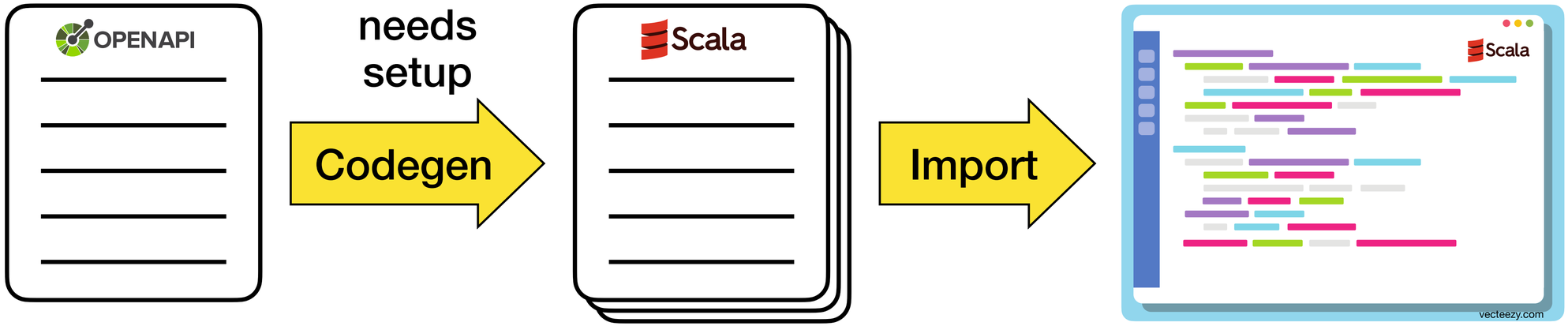
Spec-first APIs

SotA:



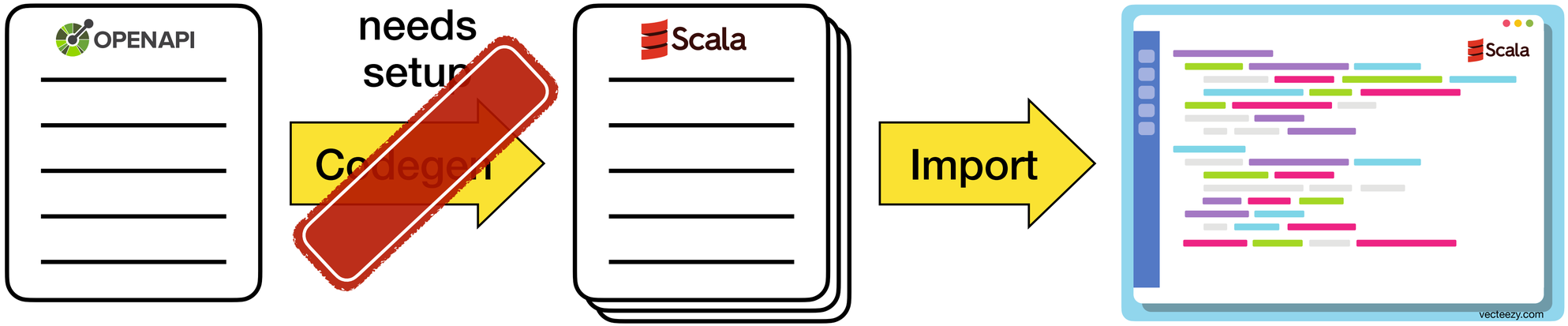
Spec-first APIs

SotA:



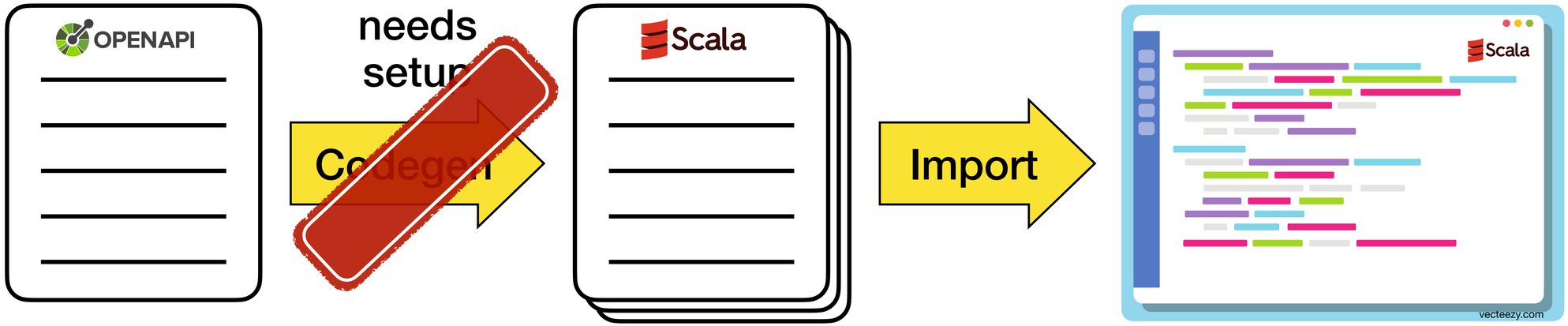
Spec-first APIs

SotA:



Spec-first APIs

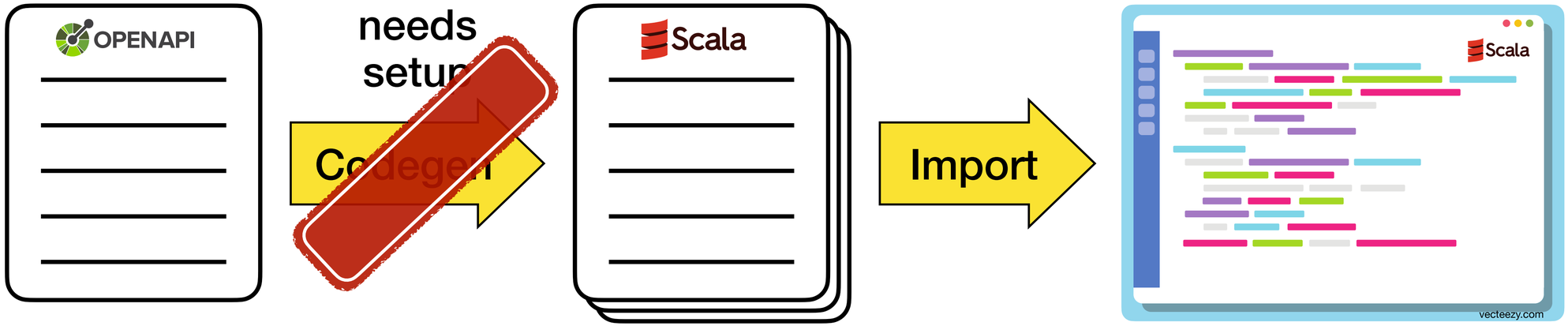
SotA:



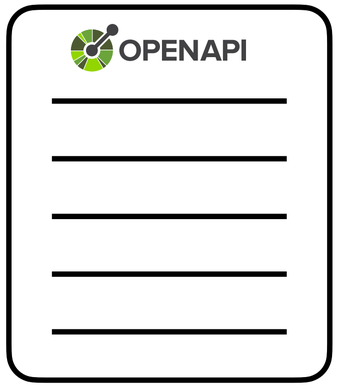
JING:

Spec-first APIs

SotA:

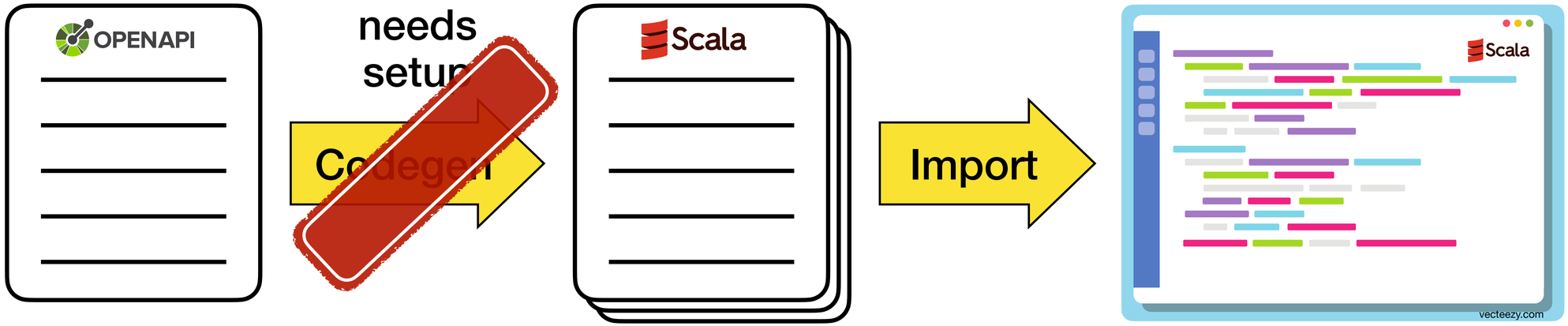


JING:

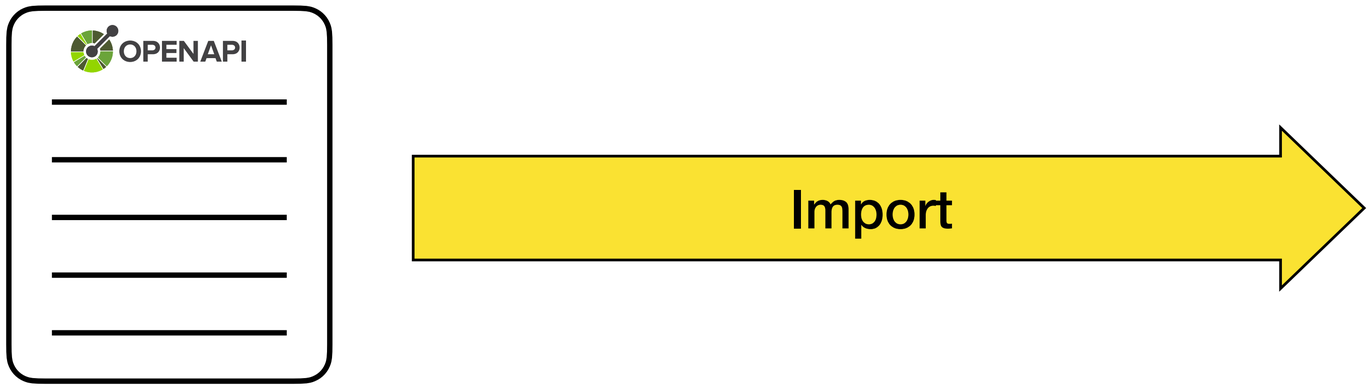


Spec-first APIs

SotA:

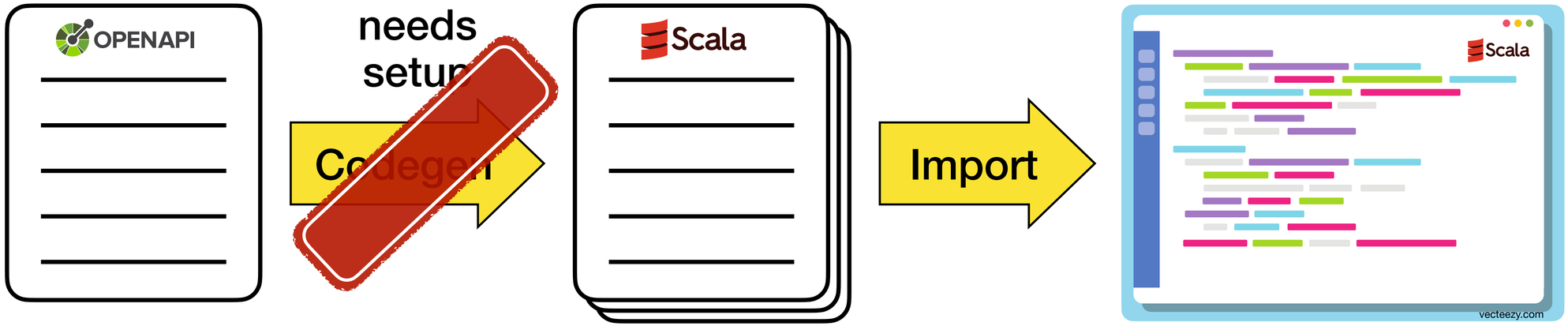


JING:

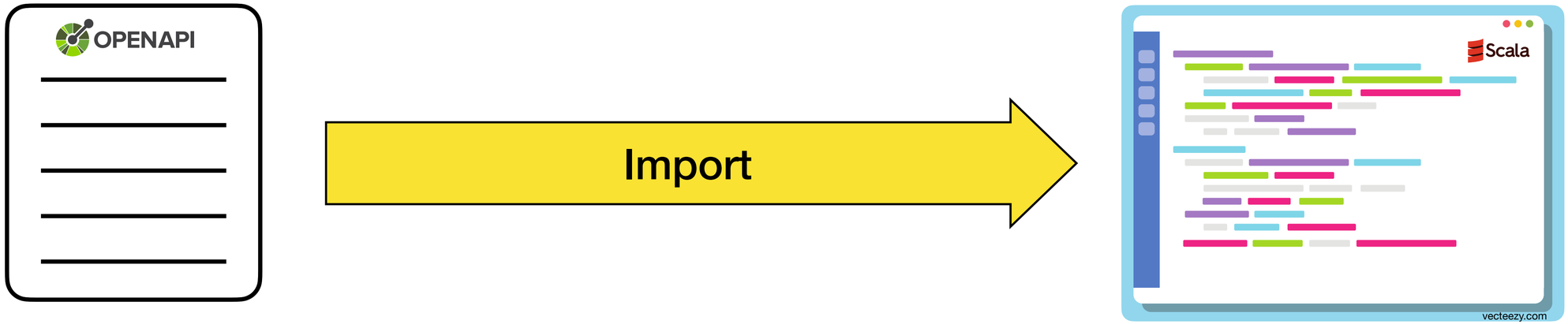


Spec-first APIs

SotA:

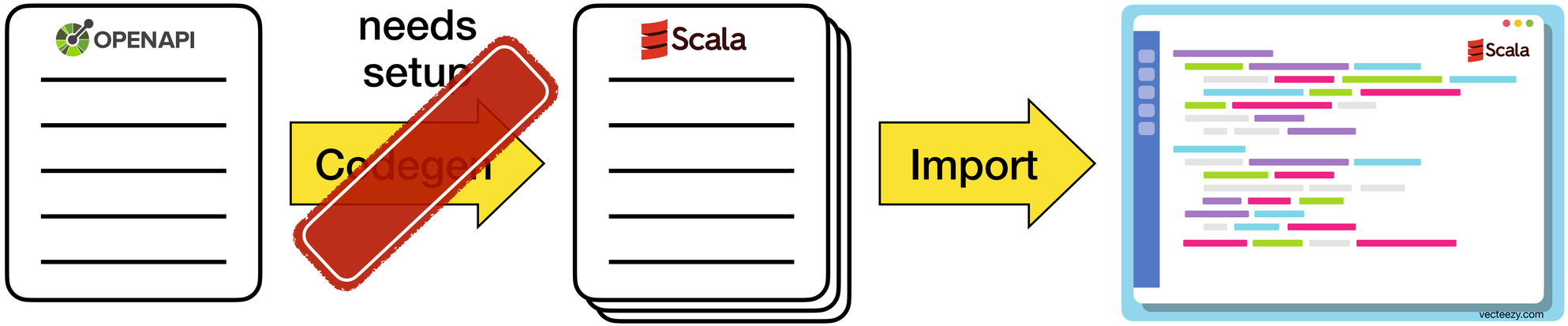


JING:

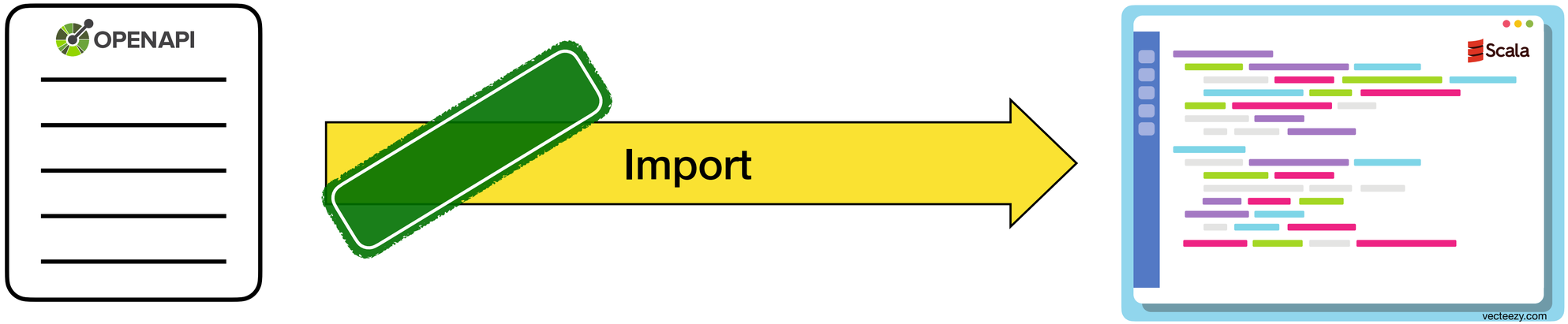


Spec-first APIs

SotA:



JING:



Demo Time!

Have I mentioned

Have I mentioned

- *Exhaustive* case and error handling

Have I mentioned

- *Exhaustive* case and error handling
  - not just the happy path

Have I mentioned

- *Exhaustive* case and error handling
  - not just the happy path
- *Type-safe*

Have I mentioned

- *Exhaustive* case and error handling
  - not just the happy path
- *Type-safe*
  - String literals *are* types

Have I mentioned

- *Exhaustive* case and error handling
  - not just the happy path
- *Type-safe*
  - String literals *are* types
- *Servers* supported

Have I mentioned

- *Exhaustive* case and error handling
  - not just the happy path
- *Type-safe*
  - String literals *are* types
- *Servers* supported
  - Incl. exhaustive endpoint handling

Have I mentioned

- *Exhaustive* case and error handling
  - not just the happy path
- *Type-safe*
  - String literals *are* types
- *Servers* supported
  - Incl. exhaustive endpoint handling
- *Localized* disruption for unsupported OpenAPI features

JING Principles Review

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

JING Principles Review

- Discoverability ✓
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

JING Principles Review

- Discoverability ✓
- Guidance ✓
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

JING Principles Review

- Discoverability ✓
- Guidance ✓
- Ramp, but no ceiling ✓
- Safety
- Hide unnecessary details
- Zero setup

JING Principles Review

- Discoverability ✓
- Guidance ✓
- Ramp, but no ceiling ✓ not feature complete
- Safety
- Hide unnecessary details
- Zero setup

JING Principles Review

- Discoverability ✓
- Guidance ✓
- Ramp, but no ceiling ✓ not feature complete
- Safety ✓
- Hide unnecessary details
- Zero setup

JING Principles Review

- Discoverability ✓
- Guidance ✓
- Ramp, but no ceiling ✓ not feature complete
- Safety ✓
- Hide unnecessary details ✓
- Zero setup

JING Principles Review

- Discoverability ✓
- Guidance ✓
- Ramp, but no ceiling ✓ not feature complete
- Safety ✓
- Hide unnecessary details ✓
- Zero setup ✓

JING Principles Review

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

JING Principles Review

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

☝ **Objection**

Applicable only to well-understood domains

JING Principles Review

- Discoverability
- Guidance
- Ramp, but no ceiling
- Safety
- Hide unnecessary details
- Zero setup

☝️ **Objection**

Applicable only to well-understood domains

🔨 **Dismissed**

Libraries, almost by definition, are for well-understood domains

Etiology

(the study of causes or origins)

Why does JING look the way it does?

Single, structurally typed value

Single, structurally typed value

```
val api = jing.openapi(“...”)
```

Single, structurally typed value

```
val api = jing.openapi("...") : OpenApiSpec {
                                  val schemas: {
                                    type Pet
                                    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
                                    // ...
                                  }
                                  val paths: {
                                    val `/pet`: {
                                      val Post: HttpEndpoint[..., ...]
                                      val Put: HttpEndpoint[..., ...]
                                      // ...
                                    }
                                    // ...
                                  }
```

Single, structurally typed value

- No generated classes/objects

```
val api = jing.openapi("...") : OpenApiSpec {
  val schemas: {
    type Pet
    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
    // ...
  }
  val paths: {
    val `/pet`: {
      val Post: HttpEndpoint[..., ...]
      val Put: HttpEndpoint[..., ...]
      // ...
    }
    // ...
  }
```

Single, structurally typed value

- No generated classes/objects
- Each type either:

```
val api = jing.openapi("...") : OpenApiSpec {
  val schemas: {
    type Pet
    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
    // ...
  }
  val paths: {
    val `/pet`: {
      val Post: HttpEndpoint[..., ...]
      val Put: HttpEndpoint[..., ...]
      // ...
    }
    // ...
  }
```

Single, structurally typed value

- No generated classes/objects
- Each type either:

  pre-defined in the library

```
val api = jing.openapi("...") : OpenApiSpec {
  val schemas: {
    type Pet
    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
    // ...
  }
  val paths: {
    val `/pet`: {
      val Post: HttpEndpoint[..., ...]
      val Put: HttpEndpoint[..., ...]
      // ...
    }
    // ...
  }
```

Single, structurally typed value

- No generated classes/objects
- Each type either:

  pre-defined in the library

  defined as type alias

```
val api = jing.openapi("...") : OpenApiSpec {
  val schemas: {
    type Pet
    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
    // ...
  }
  val paths: {
    val `/pet`: {
      val Post: HttpEndpoint[..., ...]
      val Put: HttpEndpoint[..., ...]
      // ...
    }
    // ...
  }
```

Single, structurally typed value

- No generated classes/objects
- Each type either:
  - pre-defined in the library
  - defined as type alias
  - structural refinement

```
val api = jing.openapi("...") : OpenApiSpec {
  val schemas: {
    type Pet
    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
    // ...
  }
  val paths: {
    val `/pet`: {
      val Post: HttpEndpoint[..., ...]
      val Put: HttpEndpoint[..., ...]
      // ...
    }
    // ...
  }
```

Single, structurally typed value

- No generated classes/objects
- Each type either:
  - pre-defined in the library
  - defined as type alias
  - structural refinement
- Reason:

```
val api = jing.openapi("...") : OpenApiSpec {
  val schemas: {
    type Pet
    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
    // ...
  }
  val paths: {
    val `/pet`: {
      val Post: HttpEndpoint[..., ...]
      val Put: HttpEndpoint[..., ...]
      // ...
    }
    // ...
  }
```

Single, structurally typed value

- No generated classes/objects
- Each type either:
  - pre-defined in the library
  - defined as type alias
  - structural refinement
- Reason:
  - Scala 3 macros cannot add new definitions

```
val api = jing.openapi("...") : OpenApiSpec {
  val schemas: {
    type Pet
    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
    // ...
  }
  val paths: {
    val `/pet`: {
      val Post: HttpEndpoint[..., ...]
      val Put: HttpEndpoint[..., ...]
      // ...
    }
    // ...
  }
```

Single, structurally typed value

- No generated classes/objects
- Each type either:
  - pre-defined in the library
  - defined as type alias
  - structural refinement
- **I had no choice:**
  Scala 3 macros cannot add new definitions

```
val api = jing.openapi("...") : OpenApiSpec {
  val schemas: {
    type Pet
    val Pet: ObjectSchemaCompanion[Pet, ..., ...]
    // ...
  }
  val paths: {
    val `/pet`: {
      val Post: HttpEndpoint[..., ...]
      val Put: HttpEndpoint[..., ...]
      // ...
    }
    // ...
  }
```

Why so many strange types?

Why so many strange types?

```
type Pet = Obj[
  Void
  || "id"        :? Int64
  || "name"      :: Str
  || "category"  :? Category
  || "photoUrls" :: Arr[Str]
  || "tags"      :? Arr[Tag]
  || "status"    :? Enum[Str, Void || "available" || "pending" || "sold"]]
```

Why so many strange types?

```
type Pet = Obj[
  Void
  || "id"         :? Int64
  || "name"       :: Str
  || "category"   :? Category
  || "photoUrls" :: Arr[Str]
  || "tags"       :? Arr[Tag]
  || "status"     :? Enum[Str, Void || "available" || "pending" || "sold"]]
```

Why not “simply” this?

```
type Pet = (
  id: Option[Long],
  name: String,
  category: Option[Category],
  photoUrls: Array[String],
  tags: Option[Array[Tag]],
  status: Option["available" | "pending" | "sold"],
)
```

Why so many strange types?

```
type Pet = Obj[
  Void
  || "id"         :? Int64
  || "name"       :: Str
  || "category"   :? Category
  || "photoUrls" :: Arr[Str]
  || "tags"       :? Arr[Tag]
  || "status"     :? Enum[Str, Void || "available" || "pending" || "sold"]]
```

Why not “simply” this?

```
type Pet = (
  id: Option[Long],
  name: String,
  category: Option[Category],
  photoUrls: Array[String],
  tags: Option[Array[Tag]],
  status: Option["available" | "pending" | "sold"],
)
```

1. **Faithful** domain model (optional fields, n-ary sums, base-type of enums)

Why so many strange types?

```
type Pet = Obj[
  Void
  || "id"         :? Int64
  || "name"       :: Str
  || "category"   :? Category
  || "photoUrls" :: Arr[Str]
  || "tags"       :? Arr[Tag]
  || "status"     :? Enum[Str, Void || "available" || "pending" || "sold"]]
```

Why not “simply” this?

```
type Pet = (
  id: Option[Long],
  name: String,
  category: Option[Category],
  photoUrls: Array[String],
  tags: Option[Array[Tag]],
  status: Option["available" | "pending" | "sold"],
)
```

1. **Faithful** domain model (optional fields, n-ary sums, base-type of enums)
2. **Clear** separation of Domain vs. Scala types

Why so many strange types?

```
type Pet = Obj[
  Void
  || "id"         :? Int64
  || "name"       :: Str
  || "category"   :? Category
  || "photoUrls" :: Arr[Str]
  || "tags"       :? Arr[Tag]
  || "status"     :? Enum[Str, Void || "available" || "pending" || "sold"]]
```

Why not “simply” this?

```
type Pet = (
  id: Option[Long],
  name: String,
  category: Option[Category],
  photoUrls: Array[String],
  tags: Option[Array[Tag]],
  status: Option["available" | "pending" | "sold"],
)
```

1. **Faithful** domain model (optional fields, n-ary sums, base-type of enums)
2. **Clear** separation of Domain vs. Scala types
3. Ad-hoc tuples or unions don’t make **good GADT indices**

Why so many strange types?

```
type Pet = Obj[
  Void
  || "id"         :? Int64
  || "name"       :: Str
  || "category"   :? Category
  || "photoUrls" :: Arr[Str]
  || "tags"       :? Arr[Tag]
  || "status"     :? Enum[Str, Void || "available" || "pending" || "sold"]]
```

Why not “simply” this?

```
type Pet = (
  id: Option[Long],
  name: String,
  category: Option[Category],
  photoUrls: Array[String],
  tags: Option[Array[Tag]],
  status: Option["available" | "pending" | "sold"],
)
```

I chose:

1. **Faithful** domain model (optional fields, n-ary sums, base-type of enums)
2. **Clear** separation of Domain vs. Scala types
3. Ad-hoc tuples or unions don’t make **good GADT indices**

What’s the point of `Value[_]`?

What's the point of `Value[_]`?

```
val pet = Pet(???)
```

What's the point of `Value[_]`?

```
val pet = Pet(???) : Value[Pet]
```

What's the point of `Value[_]`?

```scala
val pet = Pet(???) : Value[Pet]
```

1. To reinforce **clear separation** of Domain vs. Scala types,
   keep **Domain types uninhabited** at the Scala level.

What's the point of `Value[_]`?

```
val pet = Pet(???) : Value[Pet]
```

1. To reinforce **clear separation** of Domain vs. Scala types,
   keep **Domain types uninhabited** at the Scala level.

```
type Pet = Obj[.. || .. || ..]
```

What's the point of `Value[_]`?

```
val pet = Pet(???) : Value[Pet]
```

1. To reinforce **clear separation** of Domain vs. Scala types,
   keep **Domain types** uninhabited at the Scala level.

```
type Pet = Obj[.. || .. || ..]
```

What's the point of `Value[_]`?

```
val pet = Pet(???) : Value[Pet]
```

1. To reinforce **clear separation** of Domain vs. Scala types,
   keep **Domain types** uninhabited at the Scala level.

```
type Pet = Obj[.. || .. || ..]
```

2. **Freedom** to redefine `Value` (e.g. as a match type)
   while `Obj`, `||`, `::`, `Enum`, … remain class types (good GADT indices)

What's the point of `Value[_]`?

```
val pet = Pet(???) : Value[Pet]
```

1. To reinforce **clear separation** of Domain vs. Scala types,
   keep **Domain types** uninhabited at the Scala level.

```
type Pet = Obj[.. || .. || ..]
```

2. **Freedom** to redefine `Value` (e.g. as a match type)
   while `Obj`, `||`, `::`, `Enum`, … remain class types (good GADT indices)

```
Schema[Obj[..]]
```

What's the point of `Value[_]`?

```
val pet = Pet(???) : Value[Pet]
```

1. To reinforce **clear separation** of Domain vs. Scala types,
   keep **Domain types** uninhabited at the Scala level.

```
type Pet = Obj[.. || .. || ..]
```

2. **Freedom** to redefine `Value` (e.g. as a match type)
   while `Obj`, `||`, `::`, `Enum`, … remain class types (good GADT indices)

`Schema[Obj[..]]` ..... "index" `Obj` implies a specific case of the `Schema` ADT

What's the point of `Value[_]`?

I chose:

```
val pet = Pet(???) : Value[Pet]
```

1. To reinforce **clear separation** of Domain vs. Scala types,
   keep **Domain types** uninhabited at the Scala level.

```
type Pet = Obj[.. || .. || ..]
```

2. **Freedom** to redefine `Value` (e.g. as a match type)
   while `Obj`, `||`, `::`, `Enum`, … remain class types (good GADT indices)

`Schema[Obj[..]]` ..... "index" `Obj` implies a specific case of the `Schema` ADT

Future Work

Future Work

Future Work

Future Work

- Make feature-complete *enough*

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s
  - Skip intermediate Json objects

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s
  - Skip intermediate Json objects
    - à la Jsoniter, but parsing into `Value`s

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s
  - Skip intermediate Json objects
    - à la Jsoniter, but parsing into `Value`s
- Auto-derive transformation to a (pre-existing) Scala class

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s
  - Skip intermediate Json objects
    - à la Jsoniter, but parsing into `Value`s
- Auto-derive transformation to a (pre-existing) Scala class
  - à la Chimney or Ducktape, but transforming from `Value`s

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s
  - Skip intermediate Json objects
    - à la Jsoniter, but parsing into `Value`s
- Auto-derive transformation to a (pre-existing) Scala class
  - à la Chimney or Ducktape, but transforming from `Value`s
- Reified `Transformation`

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s
  - Skip intermediate Json objects
    - à la Jsoniter, but parsing into `Value`s
- Auto-derive transformation to a (pre-existing) Scala class
  - à la Chimney or Ducktape, but transforming from `Value`s
- Reified `Transformation`
  - `Transformation[A, Scala[B]]` *compiled* to `Array[Byte] => B | Error`

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s
  - Skip intermediate Json objects
    - à la Jsoniter, but parsing into `Value`s
- Auto-derive transformation to a (pre-existing) Scala class
  - à la Chimney or Ducktape, but transforming from `Value`s
- Reified `Transformation`
  - `Transformation[A, Scala[B]]` *compiled* to `Array[Byte] => B | Error`
  - i.e. parse directly to the domain model, skipping `Value`s

Future Work

- Make feature-complete *enough*
- Avoid excessive allocations
  - Flatter representation of `Value`s
  - Skip intermediate Json objects
    - à la Jsoniter, but parsing into `Value`s
- Auto-derive transformation to a (pre-existing)
  - à la Chimney or Ducktape, but transforming fro
- Reified `Transformation`
  - `Transformation[A, Scala[B]]` *compiled* to `Array[Byte] => B | Error`
  - i.e. parse directly to the domain model, skipping `Value`s

Takeaways

Takeaways

Zero-setup spec-first API programming?

Takeaways

Zero-setup spec-first API programming?

Hell yeah!

Takeaways

Zero-setup spec-first API programming?

Hell yeah!

JING Vision: All you need to know is Scala.

Takeaways

Zero-setup spec-first API programming?

Hell yeah!

JING Vision: All you need to know is Scala.

The rest? **Just Import 'N' Go!**

📣 Call to Action 📣

📣 Call to Action 📣

JING for

📣 Call to Action 📣

JING for

Avro

📣 Call to Action 📣

JING for

Avro

gRPC

📣 Call to Action 📣

JING for

Avro

gRPC

Smithy

📣 Call to Action 📣

JING for

Avro

gRPC

Smithy

GraphQL

📣 Call to Action 📣

JING for

Avro

gRPC

Smithy

GraphQL

AsyncAPI

📣 Call to Action 📣

JING for

Avro

gRPC

Smithy

GraphQL

AsyncAPI

...

Thank you!

github.com/TomasMikula/jing