Swag my Finch up

Get started with Finch and Swagger

Day 1

- Swagger
- Under the hood
- Endpoints
- Encoders / Decoders
- Futures
- Homework assignment

Day 2

- Testing
- Finagle Client
- Create 3rd API
- Add Authorization with Filter
- Metrics / AdminServer
- Optional: Add Security / TLS

Swagger

a.k.a Open API Specification

Swagger Specs

- Describes input and expected output of API calls
- It is described in YAML or JSON
- Language agnostic
- Both human and machine readable
- Online editor: http://editor.swagger.io/#!/
- Documentation: http://swagger.io/specification/

Swagger Specs

```yaml
swagger: '2.0'

info:
  description: This is the API for Users
  version: '1.0'
  title: Users API
  contact:
    email: you@domain.com
  license:
    name: 'Apache 2.0'
    url: 'http://www.apache.org/lincenses/LICENSE-2.0.html'
host: localhost
basePath: /v1
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /docs:
    get:
      security: []
      summary: docs
      operationId: docs
      produces:
        - application/json
      responses:
        200:
```

Task

- Paste the Documentation/ProgrammersAPI.yaml into a Swagger-Editor.
- Some parameters in ProgrammersAPI.yaml are not being used, add these parameters to the GET /programmers call
- See if you can use the error messages that are given
- Create a new endpoint to delete a Programmer by ID
- Create a new endpoint to update a Programmer by ID

Under the hood

Finagle & TwitterServer

Finch Application

```
object Main extends TwitterServer {
  val api: Service[Request, Response] = ???
  def main(): Unit = {
    val server = Http.server
      .concurrencyLimit(
         maxConcurrentRequests = 10,
         maxWaiters = 5)
      .serve(":8081", api)

    onExit { server.close() }
    Await.ready(adminHttpServer)
  }
}
```

TwitterServer

From the docs:

Always serve Finch endpoints within TwitterServer, a lightweight server template used in production at Twitter. TwitterServer wraps a **Finagle application** with a bunch of useful features such as command line flags, logging and more importantly an HTTP admin interface that can tell a lot on what’s happening with your server.

api: Service as a Function

```
abstract class Service[-Req, +Rep] extends
(Req => Future[Rep])
```

com.twitter.finagle.Http

- Server
- Client

Let's have a look

- Open the finch-workshop application in your editor.
- Go to ProgrammersAPI
- add `override def defaultHttpPort: Int = 9081` to Main object
- Start the application `sbt run`
- Fetch the list of programmers
- Fetch a specific programmer
- Add yourself as a programmer
- Postman Demo upon request

Endpoints

Matching URIs

API

- **Req**: Verb (e.g. GET) + URI + Content
- **Routing**: Req => Action
- **Action**: Req => Future[Response]
- **Controller**: Collection of Actions
- **Service**: Collection of Controllers

Finch's Endpoint[A]

- **Routing**: URI => Action
- **Action**: Req => Future[Response]

Data Flow

```
http:HTTP =>
input: Input(http) =>
Endpoint.apply(input) =>
result: EndpointResult[A] =>
Output(result) =>
http:HTTP
```

Example

```scala
val echo: Endpoint0 = "echo"
val api: Endpoint[String] = get(echo :: int) { (i: Int) =>
Ok(s"you entered $i") }

val service = api.toService

service(Request("/echo/42")) //200
service(Request("/echo/fourtytwo")) //404
```

All Endpoints are equal

- string: Endpoint[String]
- long: Endpoint[Long]
- int: Endpoint[Int]
- boolean: Endpoint[Boolean]
- uuid: Endpoint[java.lang.UUID]

- "static"

- header("foo")
- headerOption("foo")
- headerExists("foo")

- param("bar")
- paramOption("bar")
- params("bars")
- paramsNel("bars")

Parameter added

```scala
val bar = param("bar")
// '/echo/43?bar=foo'
val api: Endpoint[String] = get(echo :: int :: bar) {
  (i: Int, b: String) => Ok(s"you entered $i and $b")
}
```

Bodies

- body(Option)[A, ContentType <: String]

- jsonBody[A] == body[A, Application.Json].
- textBody[A] == body[A, Text.Plain]

Quizz

```
def patchedFoo: Endpoint[Foo => Foo] =
  jsonBody[Foo => Foo]
```

Task

- Create a "/docs" Endpoint
- Let it return a Json file

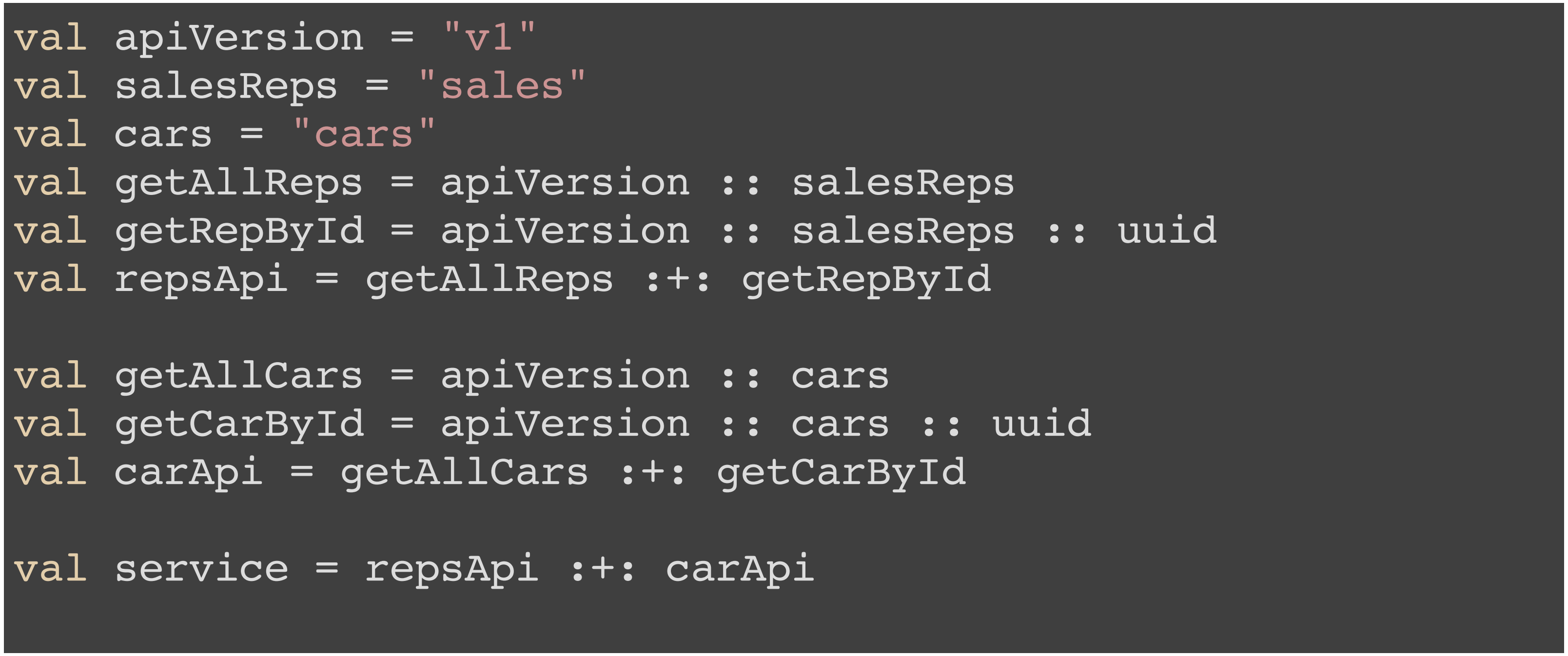
Endpoints

Composition

API

- **Action**: Req => Future[Response]
- **Routing**: Verb + URI => Action
- **Controller**: Collection of Actions
- **Service**: Collection of Controllers

AND (::) & OR (:+:)



```
val apiVersion = "v1"
val salesReps = "sales"
val cars = "cars"
val getAllReps = apiVersion :: salesReps
val getRepById = apiVersion :: salesReps :: uuid
val repsApi = getAllReps :+: getRepById

val getAllCars = apiVersion :: cars
val getCarById = apiVersion :: cars :: uuid
val carApi = getAllCars :+: getCarById

val service = repsApi :+: carApi
```

Task

- Implement the Update and Delete endpoints you've defined in Swagger
- Optional: Add a POST endpoint to create multiple programmers at once

Encoders & Decoders

Headers and Parameters

String based Endpoints

Endpoint.as[T]

```
param("foo").as[Int]
//io.finch.Endpoint[Int] = param(foo)
paramOption("bar").as[Int]
//io.finch.Endpoint[Option[Int]] = paramOption(bar)
params("bazs").as[Int]
//io.finch.Endpoint[Seq[Int]] = param(bazs)

case class Foo(i: Int, s: String)
(param("i").as[Int] :: param("s")).as[Foo]
```

Custom Decoder

```
implicit val dateTimeDecoder: DecodeEntity[DateTime] =
DecodeEntity.instance(s => Try(new DateTime(s.toLong)))

val time = param("time").as[DateTime]
val api: Endpoint[String] = get(echo :: int :: time) {
  (i: Int, b: DateTime) => Ok(s"you entered $i on $b")
}

val service = api.toService

service(Request("/echo/42?time=1489328932823")).map { x =>
  println(x.contentString)
}
```

Validation & Error Handling

```
val key = param("key")
val validkey = key.should("have length 6"){_.length > 6 }

val beValidKey = ValidationRule[String]("have length 6") { _.length > 6 }
val validkey = key.should(beValidKey)
```

- io.finch.Error.NotPresent => input failing
- io.finch.Error.NotParsed   => decoding failure
- io.finch.Error.NotValid   => validation rule fails

Task

Make an Endpoint that can respond to these requests

- calc(Request(Method.Post,"/div/42/7")) //404
- calc(Request(Method.Post,"/div/42/7?plus=five")) //400
- calc(Request(Method.Post,"/div/42/0?plus=5")) //403
- calc(Request(Method.Post,"/div/42/6?plus=5")) //200

```
//Template

post(endpoint) { (input) =>
  Ok(a / b + c)
} handle {
  case e: E =>
}
```

Encoders & Decoders

Json

JsonBody to Case Class

1. Pick a Json-lib decoder
2. Make implicit formatters available.
3. formatter => io.finch.Decode.Json[A]

http://vkostyukov.net/posts/finch-performance-lessons/

Json Decoder

finch-circe

finch-argonaut

finch-jackson

finch-json4s

finch-playjson

finch-sprayjson

Circe Json

- auto
- semiauto
- value classes

Body to Case Class

```
val fooJson = jsonBody[Foo]
val res = fooJson(Input.post("/")
           .withBody[Application.Json]
           (Buf.Utf8("""{"i":42,"s":"foo"}""")))
res.output
```

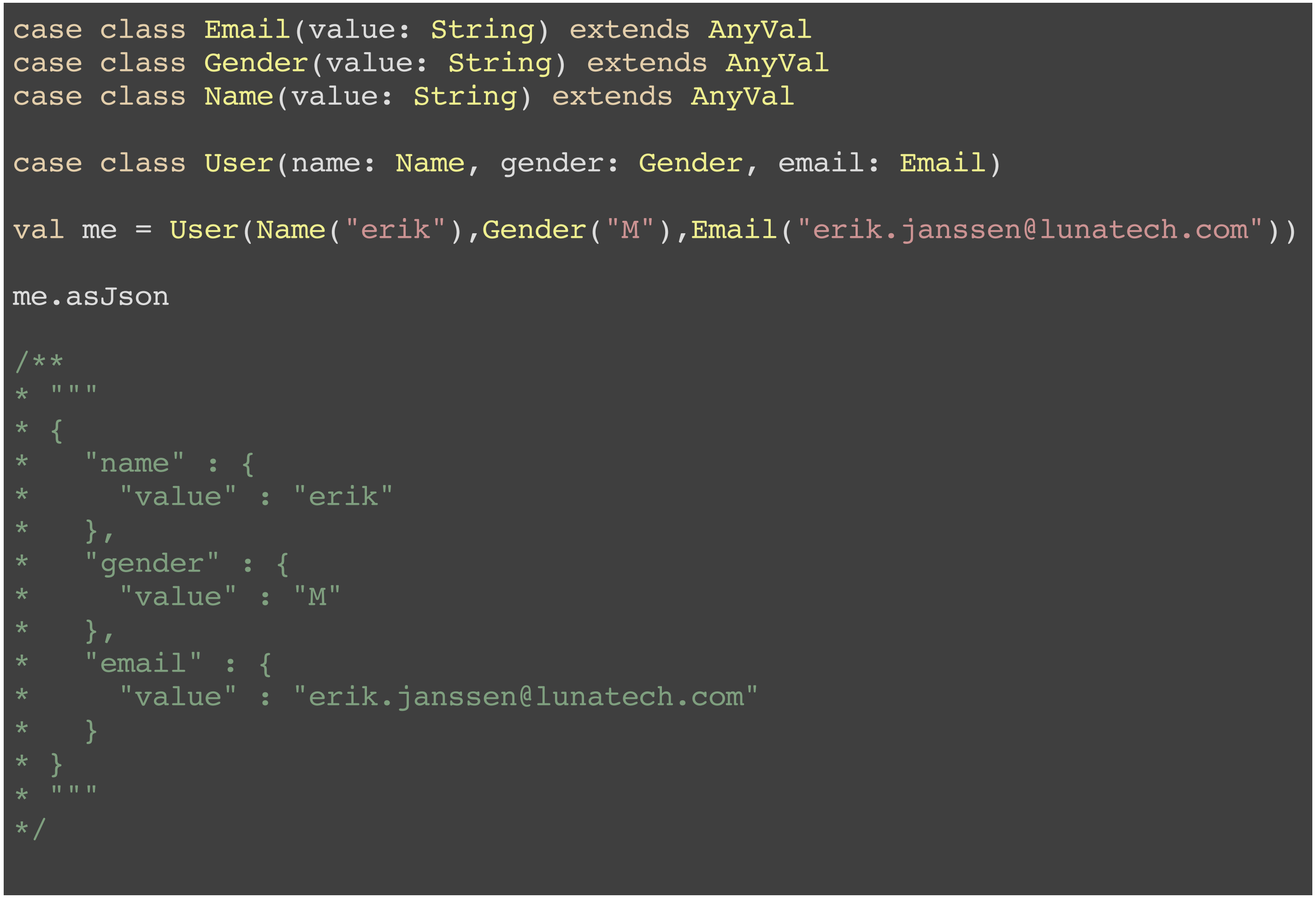
Auto

```scala
import io.circe.generic.auto._
import io.circe.syntax._

case class Foo(i: Int, s: String)

val foo = Foo(1,"s")

foo.asJson
```

Semiauto

```scala
import io.circe._
import io.circe.generic.semiauto._
import io.circe.syntax._

case class Foo(i: Int, s: String)

object Foo {
  implicit val decoder: Decoder[Foo] = deriveDecoder[Foo]
  implicit val encoder: Encoder[Foo] = deriveEncoder[Foo]
}

val foo = Foo(1,"s")
foo.asJson
```

Value Classes



```
case class Email(value: String) extends AnyVal
case class Gender(value: String) extends AnyVal
case class Name(value: String) extends AnyVal

case class User(name: Name, gender: Gender, email: Email)

val me = User(Name("erik"),Gender("M"),Email("erik.janssen@lunatech.com"))

me.asJson

/**
* """
* {
*   "name" : {
*     "value" : "erik"
*   },
*   "gender" : {
*     "value" : "M"
*   },
*   "email" : {
*     "value" : "erik.janssen@lunatech.com"
*   }
* }
* """
*/
```

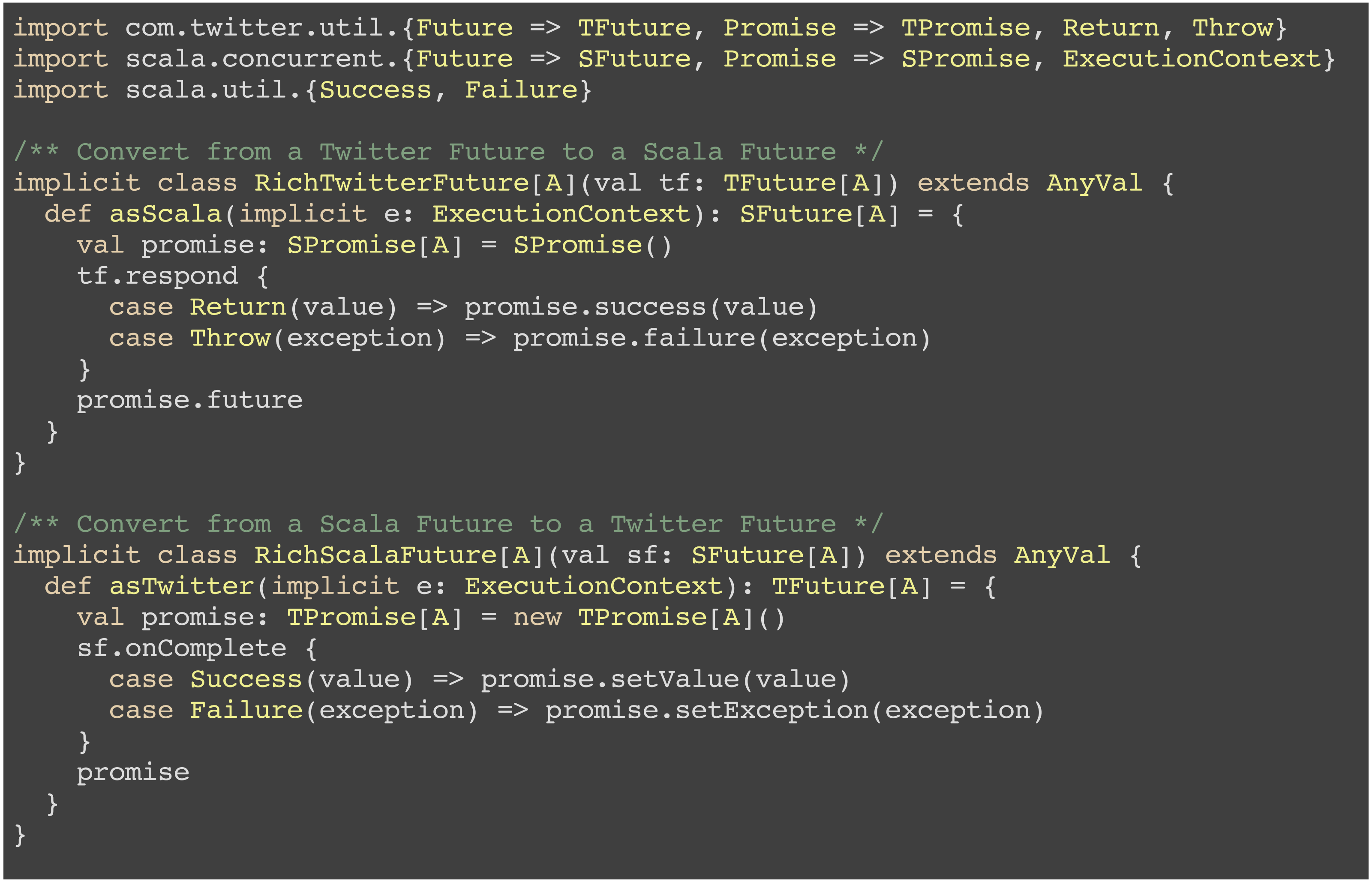
Shapeless Magic

```
import shapeless._

implicit def encoderValueClass[T <: AnyVal, V]
  (implicit g: Lazy[Generic.Aux[T, V :: HNil]],
    e: Encoder[V]): Encoder[T] = Encoder.instance {
    value ⇒
      e(g.value.to(value).head)
}

implicit def decoderValueClass[T <: AnyVal, V]
  (implicit g: Lazy[Generic.Aux[T, V :: HNil]],
   d: Decoder[V]): Decoder[T] =
   Decoder.instance {
    cursor ⇒ d(cursor).right.map { value ⇒
        g.value.from(value :: HNil)
      }
}

/**
* scala> me.asJson
* res1: io.circe.Json =
* {
*   "name" : "erik",
*   "gender" : "M",
*   "email" : "erik.janssen@lunatech.com"
* }
*
*/
```

Futures

Scala & Twitter

Conversions



```scala
import com.twitter.util.{Future => TFuture, Promise => TPromise, Return, Throw}
import scala.concurrent.{Future => SFuture, Promise => SPromise, ExecutionContext}
import scala.util.{Success, Failure}

/** Convert from a Twitter Future to a Scala Future */
implicit class RichTwitterFuture[A](val tf: TFuture[A]) extends AnyVal {
  def asScala(implicit e: ExecutionContext): SFuture[A] = {
    val promise: SPromise[A] = SPromise()
    tf.respond {
      case Return(value) => promise.success(value)
      case Throw(exception) => promise.failure(exception)
    }
    promise.future
  }
}

/** Convert from a Scala Future to a Twitter Future */
implicit class RichScalaFuture[A](val sf: SFuture[A]) extends AnyVal {
  def asTwitter(implicit e: ExecutionContext): TFuture[A] = {
    val promise: TPromise[A] = new TPromise[A]()
    sf.onComplete {
      case Success(value) => promise.setValue(value)
      case Failure(exception) => promise.setException(exception)
    }
    promise
  }
}
```

Homework

practice makes perfect

Task

- Create Skills API based on SkillsAPI.yaml file in Documentation folder
- Ping us if you have any questions

Thanks to the contributors