SwissBorg

Error Domains with Tagless Final & Cats MTL

F[_]: Ask[*[_], Context]: Raise[*[_], MyDomainError]

Overview

Note

- We'll look at this mostly from a practical perspective
- I'll avoid talking about category theory

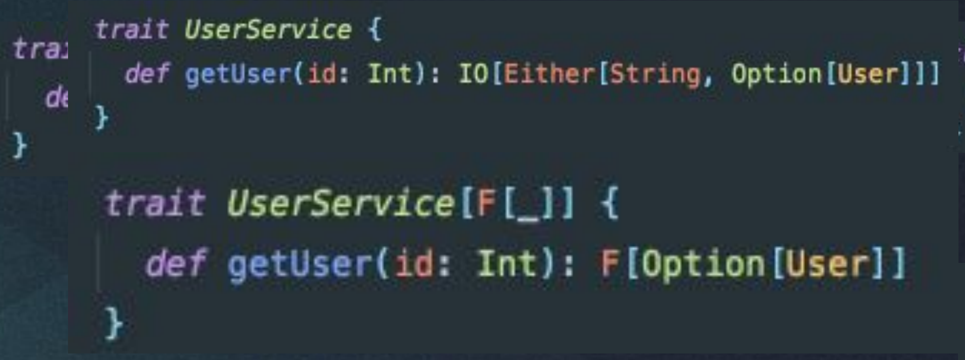
It helps if you

1. Are familiar with Typeclasses
   a. Ad hoc Polymorphism
   b. Describes a behaviour for some given type
2. Are aware of the common cats typeclasses

Overview

- Brief introduction to Tagless Final
- Monad Transformers
- Cats MTL Typeclasses
- Limiting Errors to their Domains
- FunctionK
- Sample Program
  - Running the same code in
    - IO
    - EitherT
    - Stacked EitherT/ReaderT

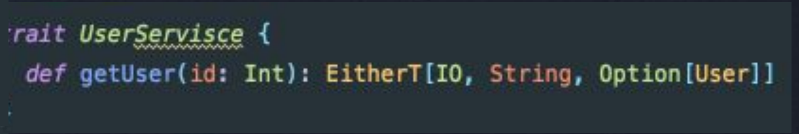
Tagless Final



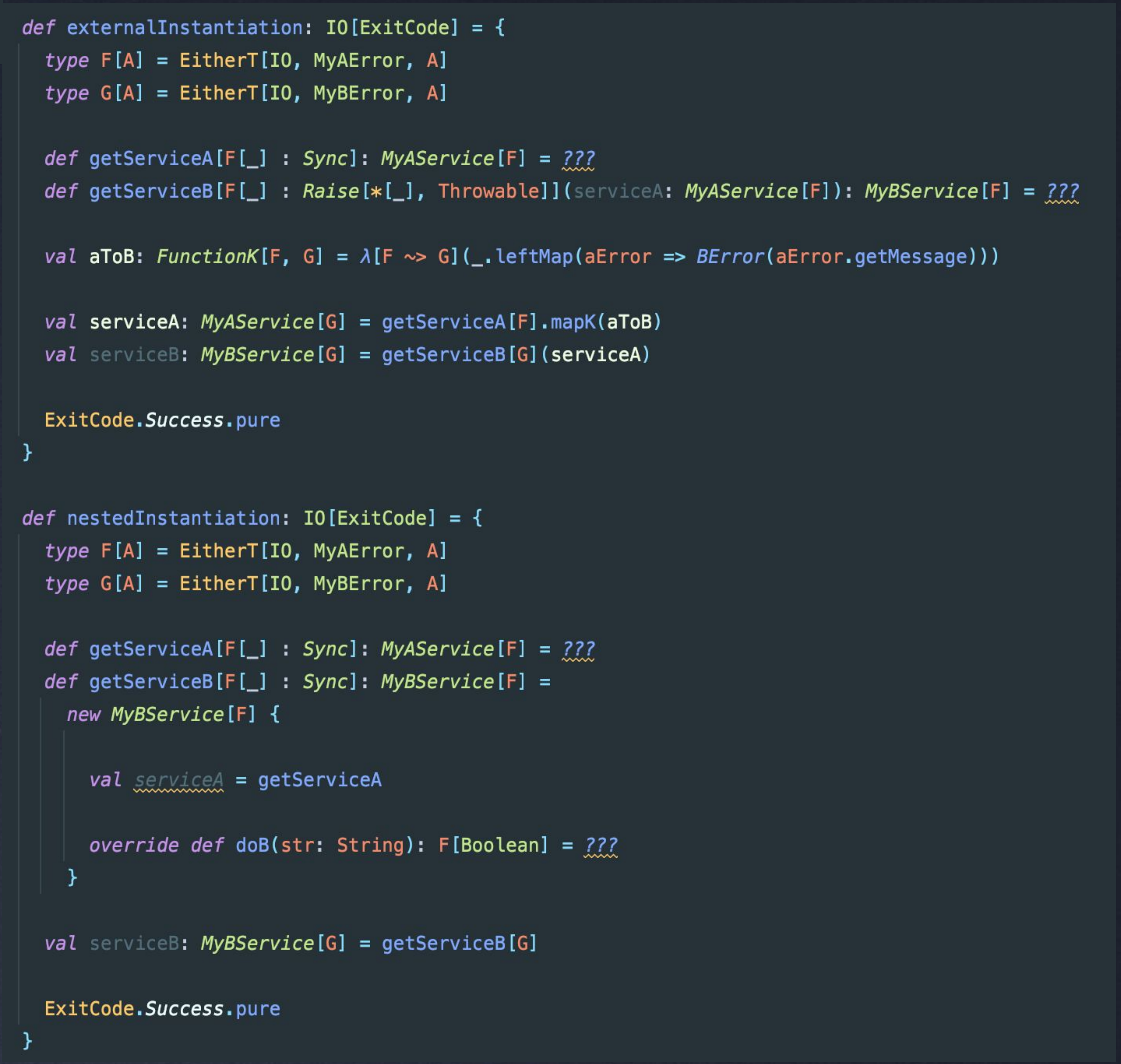
```
trai
  de
}
```

```
trait UserService {
  def getUser(id: Int): IO[Either[String, Option[User]]]
}
```

```
trait UserService[F[_]] {
  def getUser(id: Int): F[Option[User]]
}
```



```
rait UserServisce {
 def getUser(id: Int): EitherT[IO, String, Option[User]]
```



```
def externalInstantiation: IO[ExitCode] = {
  type F[A] = EitherT[IO, MyAError, A]
  type G[A] = EitherT[IO, MyBError, A]

  def getServiceA[F[_] : Sync]: MyAService[F] = ???
  def getServiceB[F[_] : Raise[*[_], Throwable]](serviceA: MyAService[F]): MyBService[F] = ???

  val aToB: FunctionK[F, G] = λ[F ~> G](_.leftMap(aError => BError(aError.getMessage)))

  val serviceA: MyAService[G] = getServiceA[F].mapK(aToB)
  val serviceB: MyBService[G] = getServiceB[G](serviceA)

  ExitCode.Success.pure
}

def nestedInstantiation: IO[ExitCode] = {
  type F[A] = EitherT[IO, MyAError, A]
  type G[A] = EitherT[IO, MyBError, A]

  def getServiceA[F[_] : Sync]: MyAService[F] = ???
  def getServiceB[F[_] : Sync]: MyBService[F] =
    new MyBService[F] {

      val serviceA = getServiceA

      override def doB(str: String): F[Boolean] = ???
    }

  val serviceB: MyBService[G] = getServiceB[G]

  ExitCode.Success.pure
}
```

Interpreter Pattern

- Code against an interface, not an implementation
  - Not specifically related to TF, but implicitly forces you to do so

Polymorphic Effect Type

- Can be instantiated in any concrete type we want that provides the required functionality

The End of the World

- Where everything that cannot be deferred happens
  - Executing, instantiating, etc.
- Implementations should be instantiated here
  - Consider the example provided
  - By instantiating MyAService inside MyBService we force ourselves to give MyAService access to Sync
    - Much more powerful than the service actually needs

Premature indirection

- Argument that TF should only be used in library code, where we don't know the Effect type that might be used.

Learning Curve

- A lot of concepts to understand

Tagless Horror - John De Goes

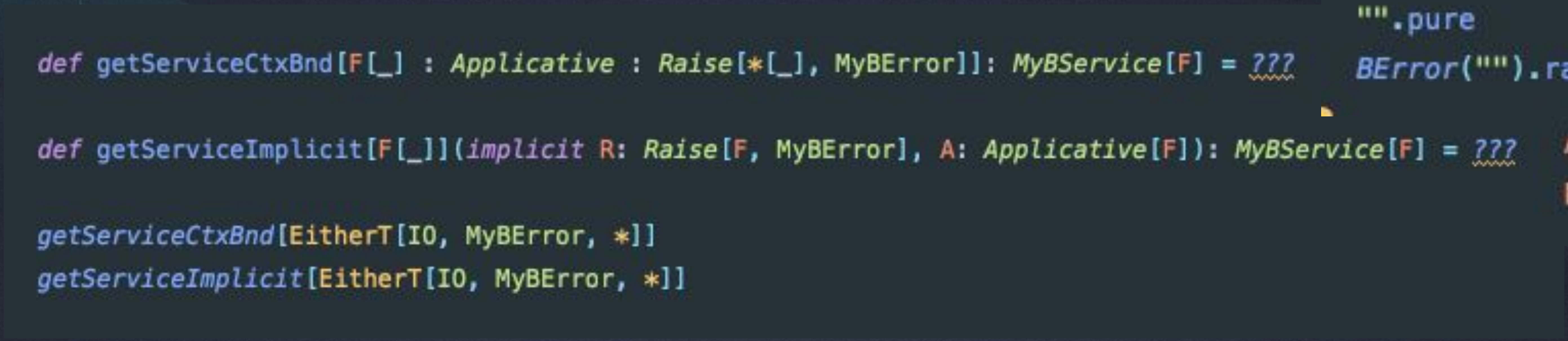
Tagless Final

Context Bounds

- Another way of defining dependencies
- Syntax sugar for implicit parameters
  - Equivalent
  - You lose the handle to the type
    - Rely on provided syntax instead
    - Also Context-Applied plugin available

Principle of least power

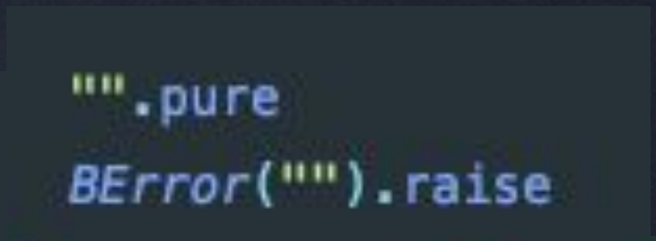
- Only need to constrain the context bounds to what your class requires
- The fewer constraints it has, the more types it supports



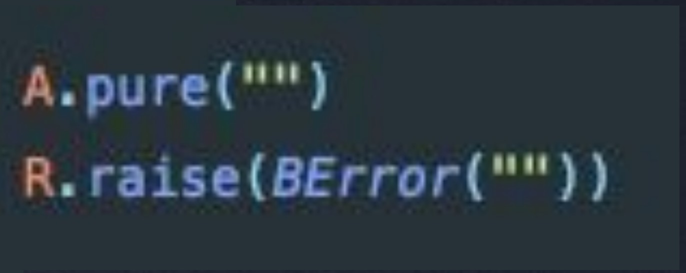
```scala
def getServiceCtxBnd[F[_] : Applicative : Raise[*[_], MyBError]]: MyBService[F] = ???

def getServiceImplicit[F[_]](implicit R: Raise[F, MyBError], A: Applicative[F]): MyBService[F] = ???

getServiceCtxBnd[EitherT[IO, MyBError, *]]
getServiceImplicit[EitherT[IO, MyBError, *]]
```



```scala
"".pure
BError("").raise
```



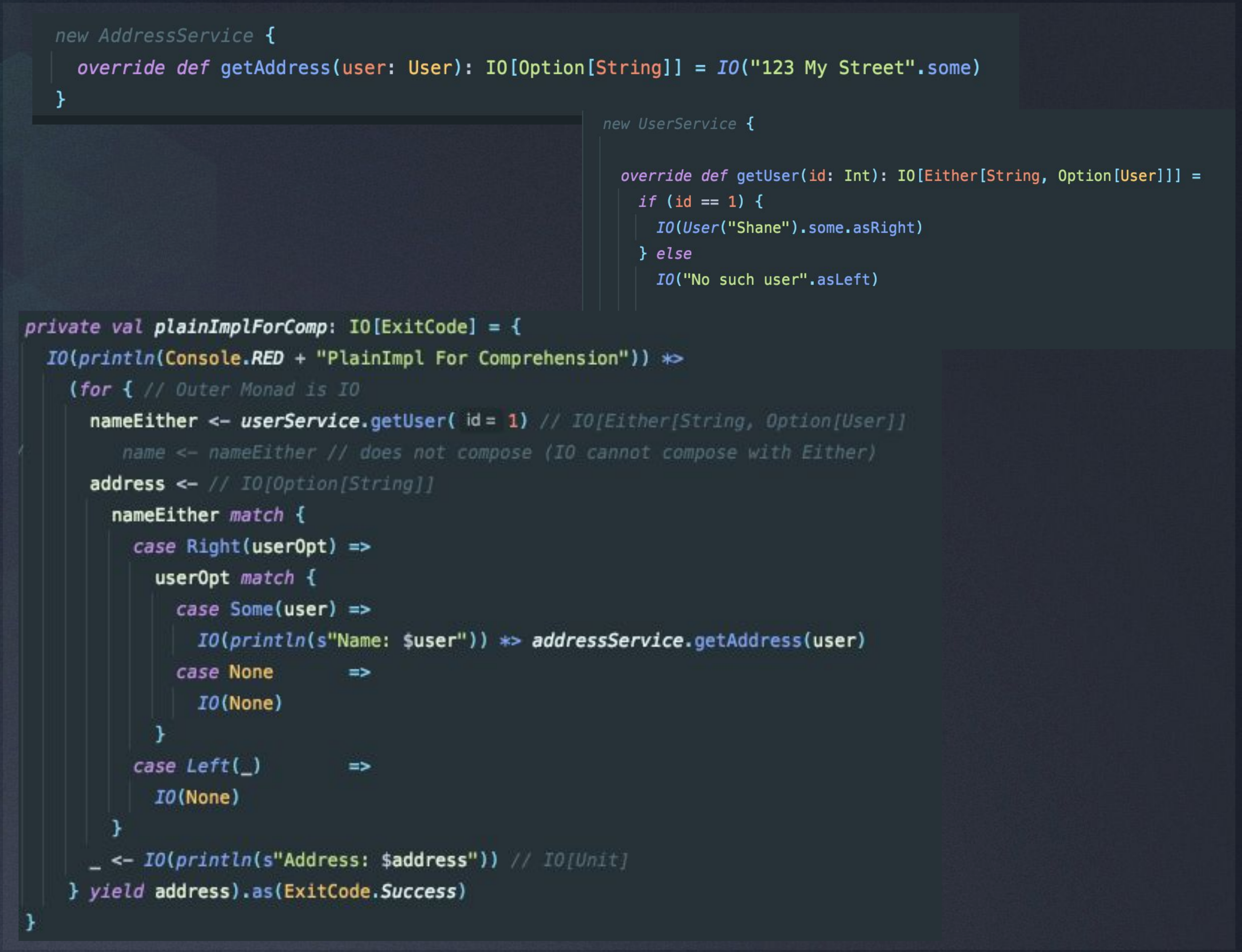
```scala
A.pure("")
R.raise(BError(""))
```

Starting out

Monads don't compose

Verbose code

- FlatMap - A => F[B]
- Manually wrangle the code into the correct monad (IO in this case)
- In order to do something simple like chain a call of
  Int => IO[Option[User]]
  To
  User => IO[String]
  We need to do a lot of wrangling



```scala
new AddressService {
  override def getAddress(user: User): IO[Option[String]] = IO("123 My Street".some)
}
```

```scala
new UserService {

  override def getUser(id: Int): IO[Either[String, Option[User]]] =
    if (id == 1) {
      IO(User("Shane").some.asRight)
    } else
      IO("No such user".asLeft)
```

```scala
private val plainImplForComp: IO[ExitCode] = {
  IO(println(Console.RED + "PlainImpl For Comprehension")) *>
    (for { // Outer Monad is IO
      nameEither <- userService.getUser( id = 1) // IO[Either[String, Option[User]]
        name <- nameEither // does not compose (IO cannot compose with Either)
      address <- // IO[Option[String]]
        nameEither match {
          case Right(userOpt) =>
            userOpt match {
              case Some(user) =>
                IO(println(s"Name: $user")) *> addressService.getAddress(user)
              case None       =>
                IO(None)
            }
          case Left(_)        =>
            IO(None)
        }
      _ <- IO(println(s"Address: $address")) // IO[Unit]
    } yield address).as(ExitCode.Success)
}
```

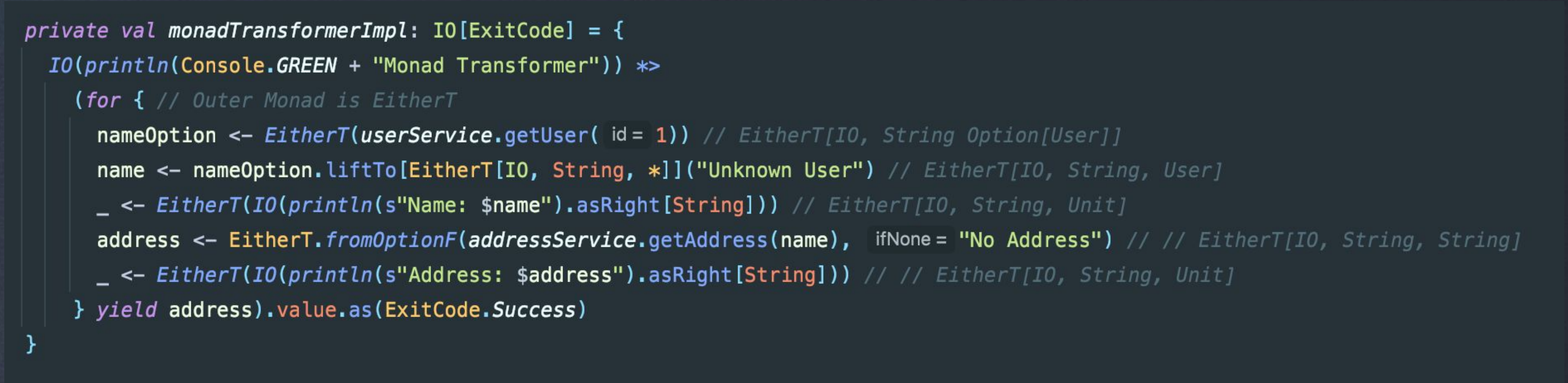
Monad Transformers

Wrapping F[_]: Monad

Monad Transformers

Effects as data types

- A wrapper around a Monad that exposes methods like `map` or `flatMap`
  - Allows us to compose because the outer Monad is the same (EitherT in this case)
- Can be difficult to work with
  - Constantly lifting values, and extracting them at the end
  - Adding new a new transformer to the stack requires changes everywhere
- This example is pretty shallow, consider how much wrangling would need to be done if we had another Monad Transformer in the stack for one of the methods
  - e.g. ReaderT[EitherT[IO, String, *], ConfigCtx, A]
- Examples: OptionT, EitherT, ReaderT



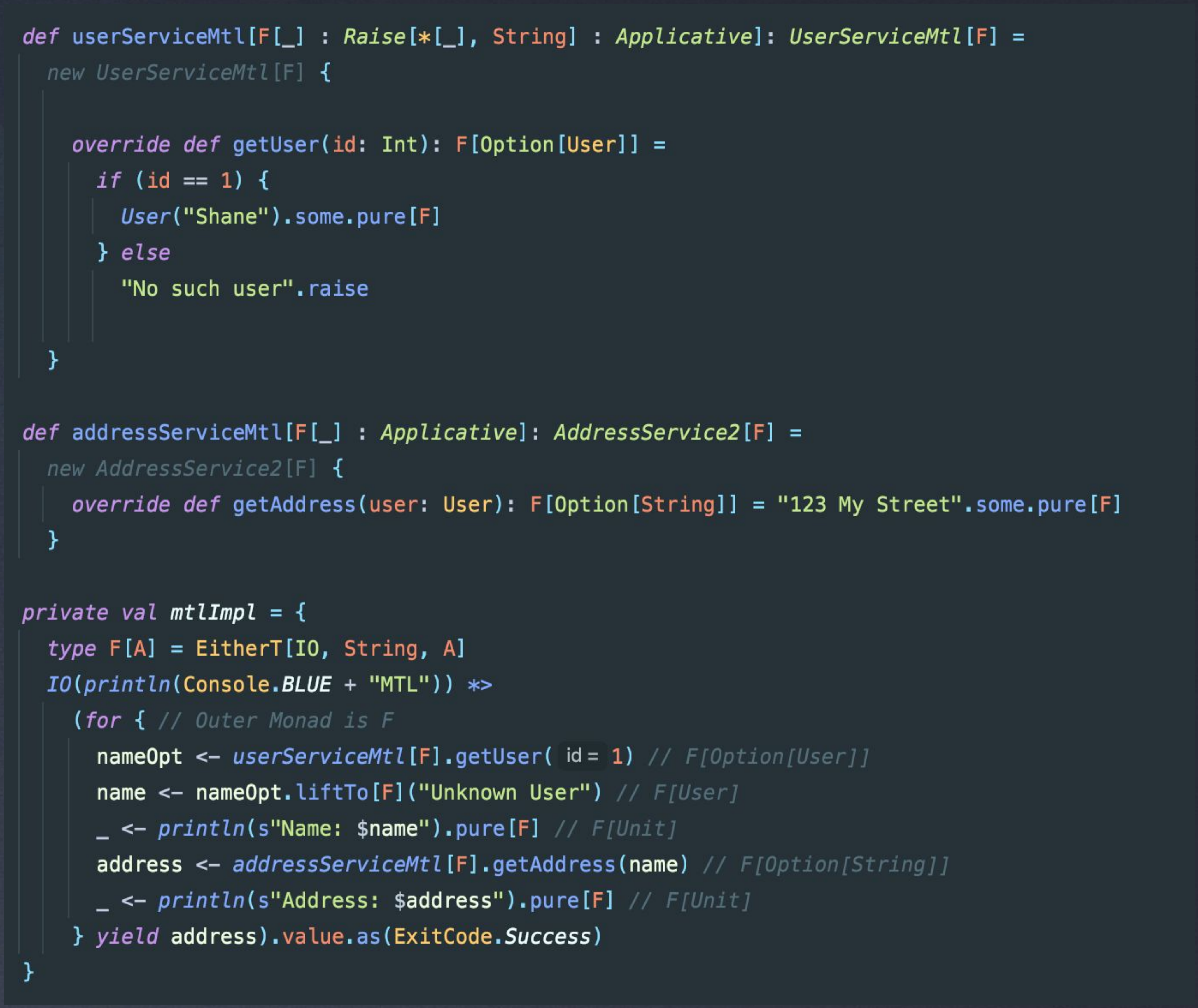
```scala
private val monadTransformerImpl: IO[ExitCode] = {
  IO(println(Console.GREEN + "Monad Transformer")) *>
    (for { // Outer Monad is EitherT
      nameOption <- EitherT(userService.getUser( id = 1)) // EitherT[IO, String Option[User]]
      name <- nameOption.liftTo[EitherT[IO, String, *]]("Unknown User") // EitherT[IO, String, User]
      _ <- EitherT(IO(println(s"Name: $name").asRight[String])) // EitherT[IO, String, Unit]
      address <- EitherT.fromOptionF(addressService.getAddress(name),  ifNone = "No Address") // // EitherT[IO, String, String]
      _ <- EitherT(IO(println(s"Address: $address").asRight[String])) // // EitherT[IO, String, Unit]
    } yield address).value.as(ExitCode.Success)
}
```

Cats MTL

Splitting F[_]
Typeclasses

Effects as Typeclasses

- Using the Tagless Final approach further, we can employ Cats MTL typeclasses to remove these Monad Transformers
- Applied as Context Bounds on our interpreters
- When instantiating our program, we give it concrete Monad datatypes (Usually Monad Transformers)
- Existing code doesn't change when we add another Transformer functionality to the stack
- Instances for most common types come for free with Cats MTL
  - WriterT, ReaderT, StateT, IorT, EitherT



```scala
def userServiceMtl[F[_] : Raise[*[_], String] : Applicative]: UserServiceMtl[F] =
  new UserServiceMtl[F] {

    override def getUser(id: Int): F[Option[User]] =
      if (id == 1) {
        User("Shane").some.pure[F]
      } else
        "No such user".raise

  }

def addressServiceMtl[F[_] : Applicative]: AddressService2[F] =
  new AddressService2[F] {
    override def getAddress(user: User): F[Option[String]] = "123 My Street".some.pure[F]
  }

private val mtlImpl = {
  type F[A] = EitherT[IO, String, A]
  IO(println(Console.BLUE + "MTL")) *>
    (for { // Outer Monad is F
      nameOpt <- userServiceMtl[F].getUser( id = 1) // F[Option[User]]
      name <- nameOpt.liftTo[F]("Unknown User") // F[User]
      _ <- println(s"Name: $name").pure[F] // F[Unit]
      address <- addressServiceMtl[F].getAddress(name) // F[Option[String]]
      _ <- println(s"Address: $address").pure[F] // F[Unit]
    } yield address).value.as(ExitCode.Success)
}
```

Typeclasses

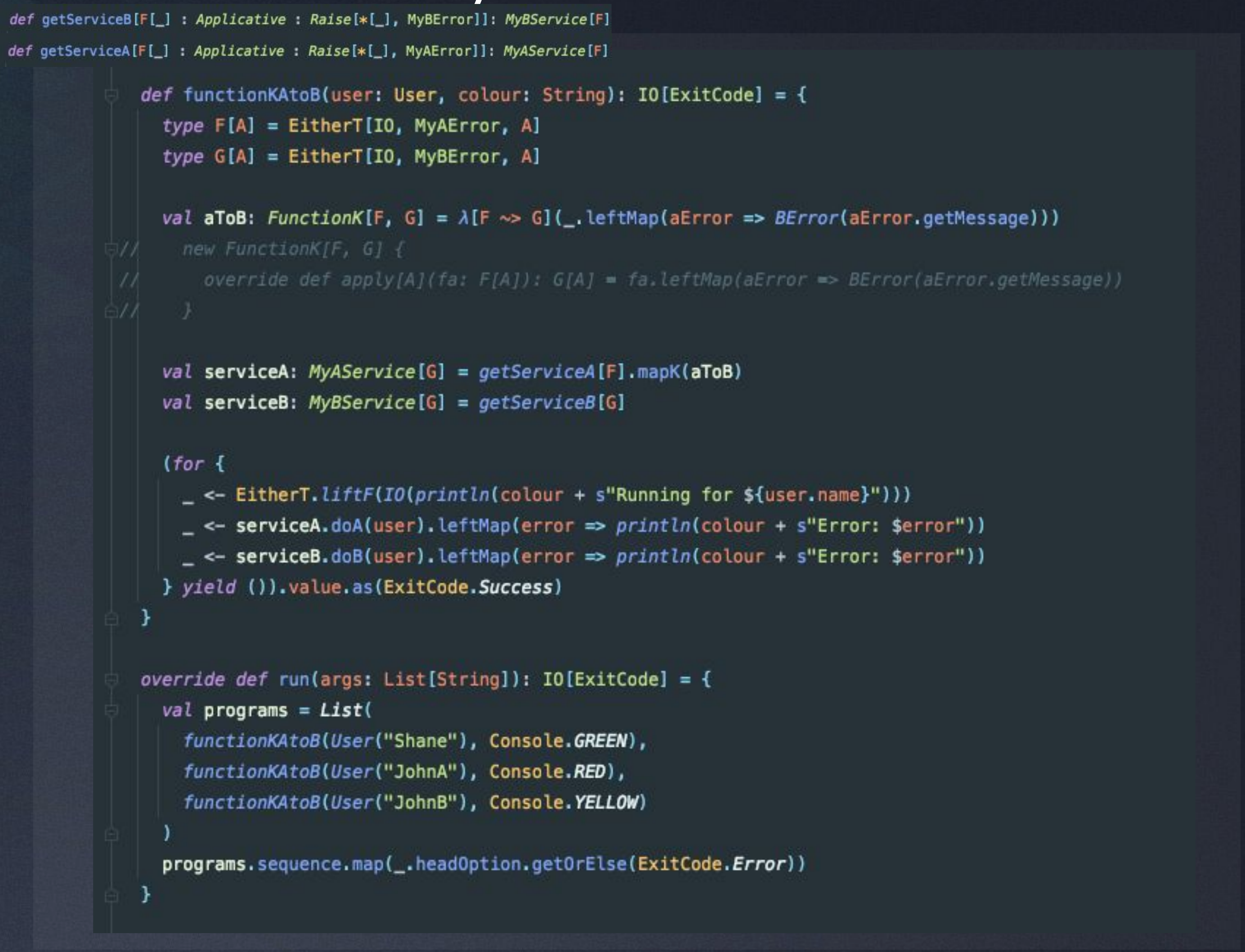
Typeclass	Signature	Library	Represents	Example Type
MonadError	MonadError[F, E]	Cats	Ability to Raise and Handle Errors	**Either**
ApplicativeError	ApplicativeError[F, E]	Cats	Ability to Raise and Handle Errors	**Validated**
Raise	Raise[F, E]	Cats MTL	Ability to Raise Errors	**Either**
Handle	Handle[F, E]	Cats MTL	Ability to Raise and Handle Errors	**Either**
Kleisli	Kleisli[F, A, B]	Cats	Function of A => F[B]	**ReaderT**
ReaderT	ReaderT[F, A, B]	Cats	Type Alias for Kleisli	**Kleisli**
Ask	Ask[F, E]	Cats MTL	Ability to ask for a value from our Environment	**Kleisli**
Local	Local[F, E]	Cats MTL	Ability to Locally Modify our Environment	**Kleisli**
FunctionK	**FunctionK[F, G]**	Cats	Expresses the Transformation for F ~> G	**FunctionK (Data Type)**

Why not use the same Monad everywhere?

Domain specific Errors
F[_]: Raise[*[_], My*Error]

MyDBError vs MyServiceError

- If we want to keep our Error domains isolated
  - How can we compose EitherT[IO, MyDBError, A] with EitherT[IO, MyServiceError, A]?
- We need to translate this somehow
- FunctionK[F, G] / F ~> G
  - A function from one higher kinded F[_] type to another (F ~> G)
  - E.g. EitherT[IO, MyDBError, A] ~> EitherT[IO, MyServiceError, A]
- Using an ADT for our Error datatypes forces us to handle each Error type of our domain and map it to the next domain level



```scala
def getServiceB[F[_] : Applicative : Raise[*[_], MyBError]]: MyBService[F]
def getServiceA[F[_] : Applicative : Raise[*[_], MyAError]]: MyAService[F]

def functionKAtoB(user: User, colour: String): IO[ExitCode] = {
  type F[A] = EitherT[IO, MyAError, A]
  type G[A] = EitherT[IO, MyBError, A]

  val aToB: FunctionK[F, G] = λ[F ~> G](_.leftMap(aError => BError(aError.getMessage)))
//  new FunctionK[F, G] {
//    override def apply[A](fa: F[A]): G[A] = fa.leftMap(aError => BError(aError.getMessage))
//  }

  val serviceA: MyAService[G] = getServiceA[F].mapK(aToB)
  val serviceB: MyBService[G] = getServiceB[G]

  (for {
    _ <- EitherT.liftF(IO(println(colour + s"Running for ${user.name}")))
    _ <- serviceA.doA(user).leftMap(error => println(colour + s"Error: $error"))
    _ <- serviceB.doB(user).leftMap(error => println(colour + s"Error: $error"))
  } yield ()).value.as(ExitCode.Success)
}

override def run(args: List[String]): IO[ExitCode] = {
  val programs = List(
    functionKAtoB(User("Shane"), Console.GREEN),
    functionKAtoB(User("JohnA"), Console.RED),
    functionKAtoB(User("JohnB"), Console.YELLOW)
  )
  programs.sequence.map(_.headOption.getOrElse(ExitCode.Error))
}
```

Sample Program

Overview

- The sample application demonstrates combining 3 distinct error domains
- It exposes an API, which when called will
  - Take input provided by the caller
  - Combine this with some output obtained from another external service (FOAAS)
  - Save the result to the database
- We try to demonstrate the distinct Error domains with 3 Error examples
  - The names provided via the API must begin with a capital letter
  - The DB column is VARCHAR(200) and will fail to insert anything longer
    - This should probably be a technical error rather than a DB Domain error, but for illustration purposes…
    - Maybe you want to handle this “domain” error by truncating the input or something
  - The External API response cannot be decoded
- For each of these errors, we expect a different Status code response to be returned from the API

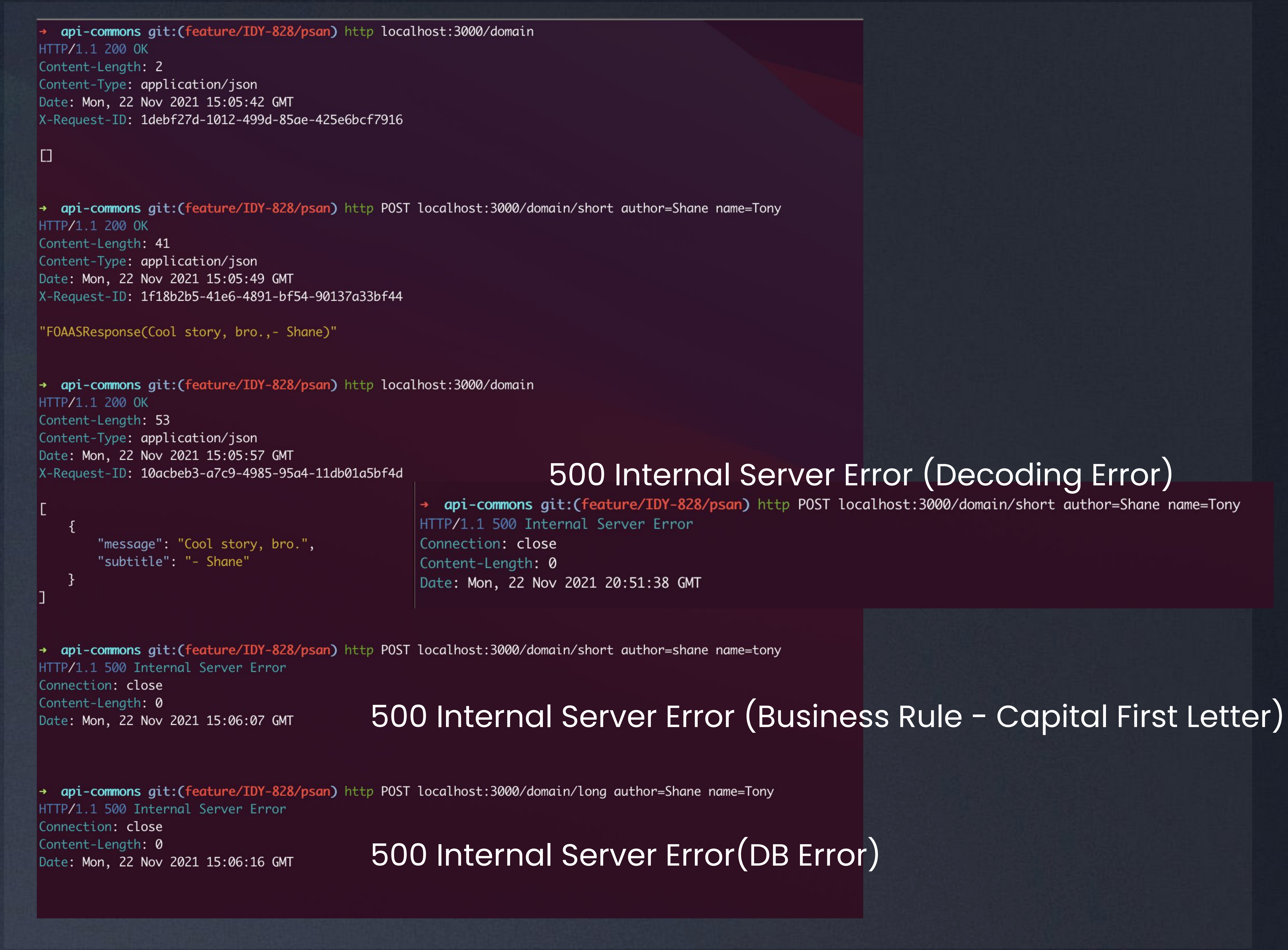
Domain Error Name	Service Error Name	Description	**Status Code**
DecodingError	ExternalCallGenericError	All External Calls are mapped to a generic error and reported in the API as a 503 Error	**503 Internal Service Error**
TooLongError	InputTooLargeError	SQL Errors with a state code of 22001 are reported as TooLongErrors and reported in the API as 413 Errors, all other DB errors are reported as 503 Errors	**413 Payload Too Large**
NameFormatError	NameFormatError	Validation is performed on the API input, if either name is not Capitalised, we return a 400 Error in the API	**400 Bad Request**

Sample Program

IO

- The whole app is in the “Run” context, which is an alias for IO
- Because IO doesn’t provide a typeclass instance for Raise (only Raise for Throwable is provided), we provide our own naive implementation
  - Raise[Run, E]
  - Raise an exception with the domainError.toString as the message
- We lose the ability to distinguish the errors and handle them appropriately in our HttpApp, since they just get raise as an IO failure
- Http4s handles these exceptions by default with a 500 error

- If our Domain Errors extended Throwable, we could have provided a nicer implementation here, but would have lost the possibility to ensure we exhaustively handled all our domain errors



```
→  api-commons git:(feature/IDY-828/psan) http localhost:3000/domain
HTTP/1.1 200 OK
Content-Length: 2
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:05:42 GMT
X-Request-ID: 1debf27d-1012-499d-85ae-425e6bcf7916

[]


→  api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=Shane name=Tony
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:05:49 GMT
X-Request-ID: 1f18b2b5-41e6-4891-bf54-90137a33bf44

"FOAASResponse(Cool story, bro.,- Shane)"


→  api-commons git:(feature/IDY-828/psan) http localhost:3000/domain
HTTP/1.1 200 OK
Content-Length: 53
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:05:57 GMT
X-Request-ID: 10acbeb3-a7c9-4985-95a4-11db01a5bf4d

[
    {
        "message": "Cool story, bro.",
        "subtitle": "- Shane"
    }
]
```

500 Internal Server Error (Decoding Error)

```
→  api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=Shane name=Tony
HTTP/1.1 500 Internal Server Error
Connection: close
Content-Length: 0
Date: Mon, 22 Nov 2021 20:51:38 GMT
```

```
→  api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=shane name=tony
HTTP/1.1 500 Internal Server Error
Connection: close
Content-Length: 0
Date: Mon, 22 Nov 2021 15:06:07 GMT
```

500 Internal Server Error (Business Rule - Capital First Letter)

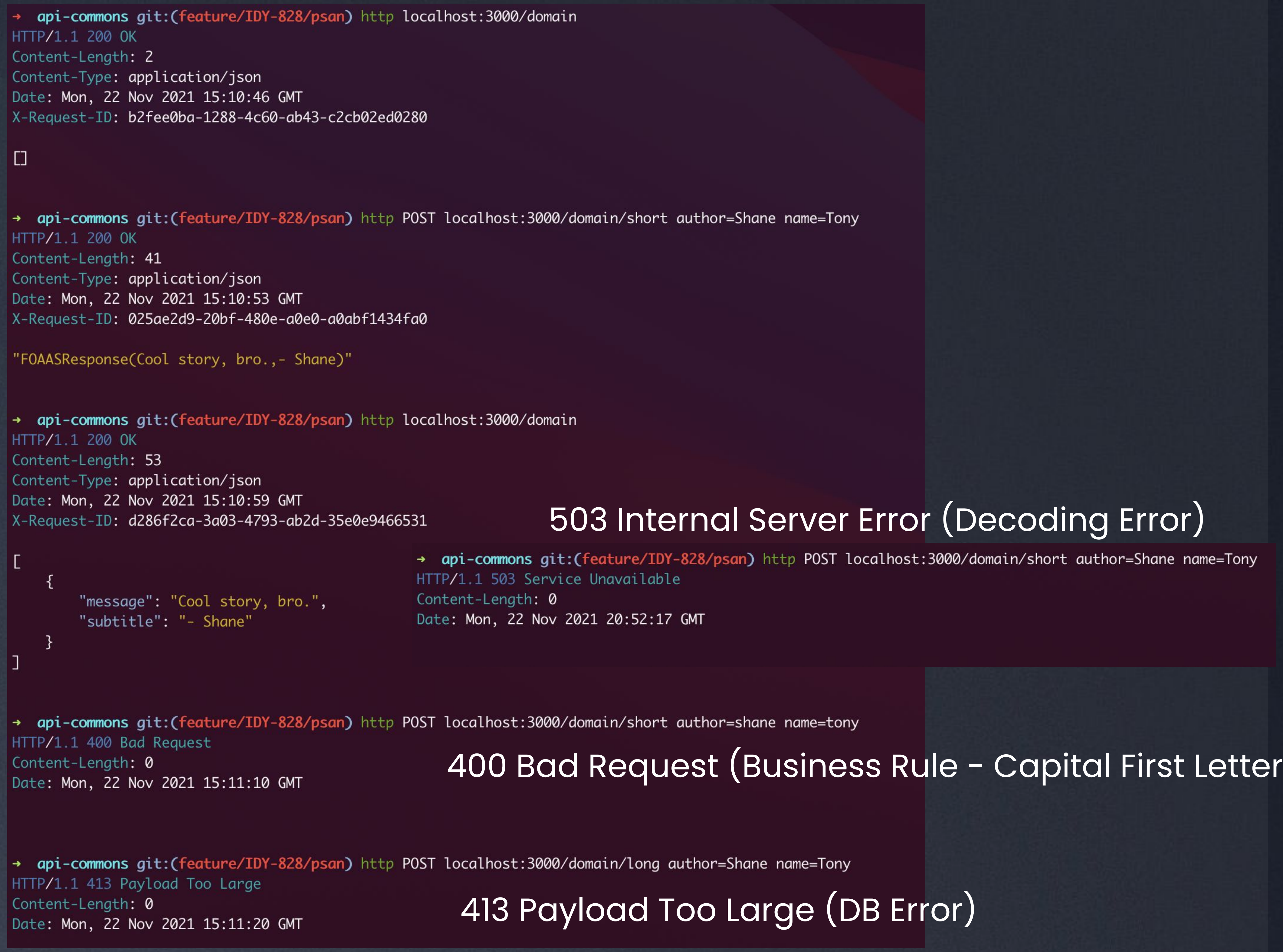
```
→  api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/long author=Shane name=Tony
HTTP/1.1 500 Internal Server Error
Connection: close
Content-Length: 0
Date: Mon, 22 Nov 2021 15:06:16 GMT
```

500 Internal Server Error(DB Error)

Sample Program

EitherT

- Each Service has it's own Error Domain
  - HttpClient
  - Database
  - Service
- We get our Raise and Handle instances for free (derived from ApplicativeError instance provided for EitherT)
- Since each domain operates in it's own error context, we need to "translate" each error domain explicitly (using FunctionK)
- This allows us to map our domain errors to the correct HTTP response codes



```
→ api-commons git:(feature/IDY-828/psan) http localhost:3000/domain
HTTP/1.1 200 OK
Content-Length: 2
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:10:46 GMT
X-Request-ID: b2fee0ba-1288-4c60-ab43-c2cb02ed0280

[]


→ api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=Shane name=Tony
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:10:53 GMT
X-Request-ID: 025ae2d9-20bf-480e-a0e0-a0abf1434fa0

"FOAASResponse(Cool story, bro.,- Shane)"


→ api-commons git:(feature/IDY-828/psan) http localhost:3000/domain
HTTP/1.1 200 OK
Content-Length: 53
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:10:59 GMT
X-Request-ID: d286f2ca-3a03-4793-ab2d-35e0e9466531

[
    {
        "message": "Cool story, bro.",
        "subtitle": "- Shane"
    }
]
```

503 Internal Server Error (Decoding Error)

```
→ api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=Shane name=Tony
HTTP/1.1 503 Service Unavailable
Content-Length: 0
Date: Mon, 22 Nov 2021 20:52:17 GMT
```

```
→ api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=shane name=tony
HTTP/1.1 400 Bad Request
Content-Length: 0
Date: Mon, 22 Nov 2021 15:11:10 GMT
```

400 Bad Request (Business Rule - Capital First Letter)

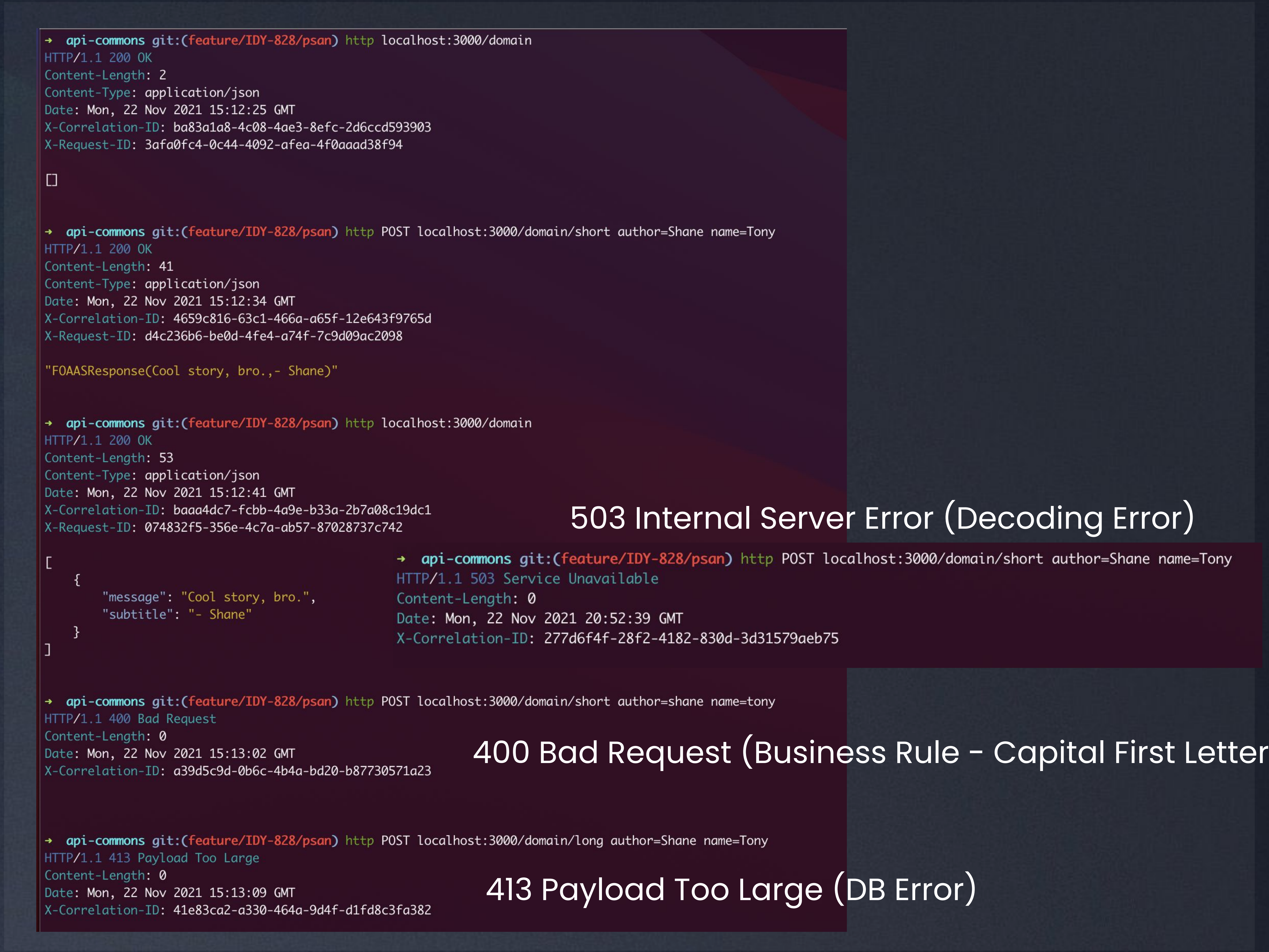
```
→ api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/long author=Shane name=Tony
HTTP/1.1 413 Payload Too Large
Content-Length: 0
Date: Mon, 22 Nov 2021 15:11:20 GMT
```

413 Payload Too Large (DB Error)

Sample Program

ReaderT

- Each Service has it’s own Error Domain
    - HttpClient
    - Database
    - Service
- We get our Raise and Handle instances for free (derived from ApplicativeError instance provided for EitherT)
- Our Logger is aware that our effect may contain a Context
    - If the context exists, it logs it
- Once we provide this, it gets logged out by our logger



```
→  api-commons git:(feature/IDY-828/psan) http localhost:3000/domain
HTTP/1.1 200 OK
Content-Length: 2
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:12:25 GMT
X-Correlation-ID: ba83a1a8-4c08-4ae3-8efc-2d6ccd593903
X-Request-ID: 3afa0fc4-0c44-4092-afea-4f0aaad38f94

[]


→  api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=Shane name=Tony
HTTP/1.1 200 OK
Content-Length: 41
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:12:34 GMT
X-Correlation-ID: 4659c816-63c1-466a-a65f-12e643f9765d
X-Request-ID: d4c236b6-be0d-4fe4-a74f-7c9d09ac2098

"FOAASResponse(Cool story, bro.,- Shane)"


→  api-commons git:(feature/IDY-828/psan) http localhost:3000/domain
HTTP/1.1 200 OK
Content-Length: 53
Content-Type: application/json
Date: Mon, 22 Nov 2021 15:12:41 GMT
X-Correlation-ID: baaa4dc7-fcbb-4a9e-b33a-2b7a08c19dc1
X-Request-ID: 074832f5-356e-4c7a-ab57-87028737c742

[
    {
        "message": "Cool story, bro.",
        "subtitle": "- Shane"
    }
]


→  api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=shane name=tony
HTTP/1.1 400 Bad Request
Content-Length: 0
Date: Mon, 22 Nov 2021 15:13:02 GMT
X-Correlation-ID: a39d5c9d-0b6c-4b4a-bd20-b87730571a23


→  api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/long author=Shane name=Tony
HTTP/1.1 413 Payload Too Large
Content-Length: 0
Date: Mon, 22 Nov 2021 15:13:09 GMT
X-Correlation-ID: 41e83ca2-a330-464a-9d4f-d1fd8c3fa382
```

503 Internal Server Error (Decoding Error)

```
→  api-commons git:(feature/IDY-828/psan) http POST localhost:3000/domain/short author=Shane name=Tony
HTTP/1.1 503 Service Unavailable
Content-Length: 0
Date: Mon, 22 Nov 2021 20:52:39 GMT
X-Correlation-ID: 277d6f4f-28f2-4182-830d-3d31579aeb75
```

400 Bad Request (Business Rule - Capital First Letter)

413 Payload Too Large (DB Error)

Questions?

Where's the code?

- https://github.com/smur89/scala_domain_errors

SwissBorg

Headquartered in Switzerland:

SBorgSA
Rue du Grand-Cheêne 8, 1003 Lausanne, Switzerland

Regulated in Europe:

SwissBorg Solutions OÜ
Roosikrantsi tn 2-1091,Kesklinna linnaosa, Tallinn, Harju maakond 10119, Estonia

Licenses:

SwissBorg Solutions OÜ is the holder of Virtual Currency License number FVT000326. For more details, please see our Terms of Use.

Proudly made in Switzerland

www.swissborg.com