SCALA AT MATYGO

Friday, 4 May, 12

MY NAME IS JOE

BLOG: JOEGAUDET.TUMBLR.COM
TWITTER: @JOGAUDET
GITHUB: GITHUB.COM/JOEGAUDET

Who am I and why should you listen to me? Well I am joe, this is where you find me on the internet.

CTO AT MATYGO

I’m a founder at matygo, I’ve been working as the CTO for 2 years now.

BSCENG & (.95)MASC

Friday, 4 May, 12

I hold a BScEng in Computer Engineering from UNB and am almost *sigh* done a MASC of Computer Engineering from UBC.

GNAR ENTHUSIAST

Friday, 4 May, 12

I love skiing.

BREWMASTER

Friday, 4 May, 12

And making beer.

PROGRAMMER

But most importantly I am a programmer, I've been hacking away since I was a kid. But really first started doing web stuff in PHP. I've worked extensively in Oil and Gas stuff with Java. I've done some low level embedded C work. Piles of Rails stuff, but most recently JS in the form of Sproutcore and Scala.

MATYGO

WE BUILD LEARNING TOOLS

Friday, 4 May, 12

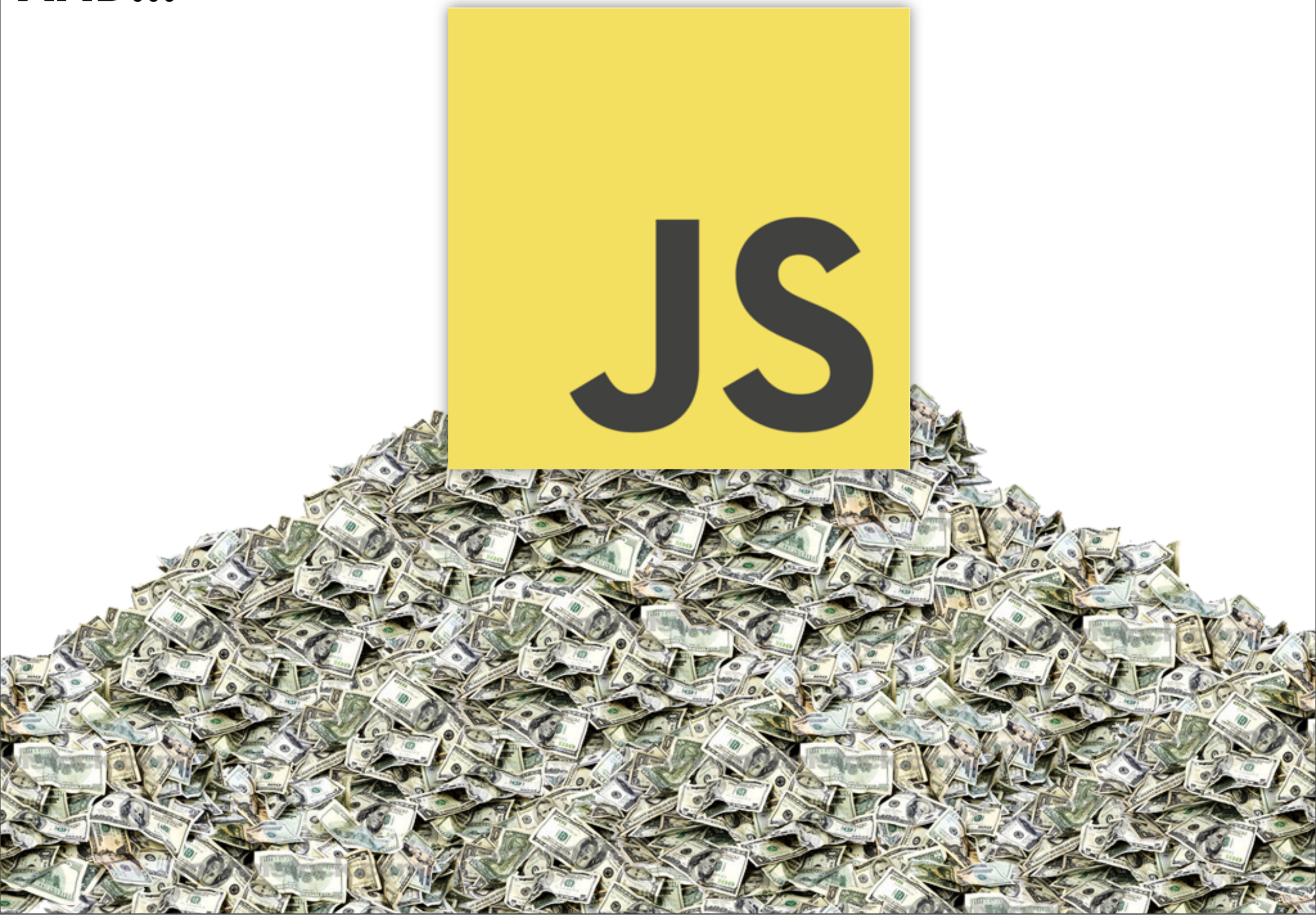
At matygo we build tools to help deliver learning

- In the corporate training
- Higher Ed
- Or private small group learning

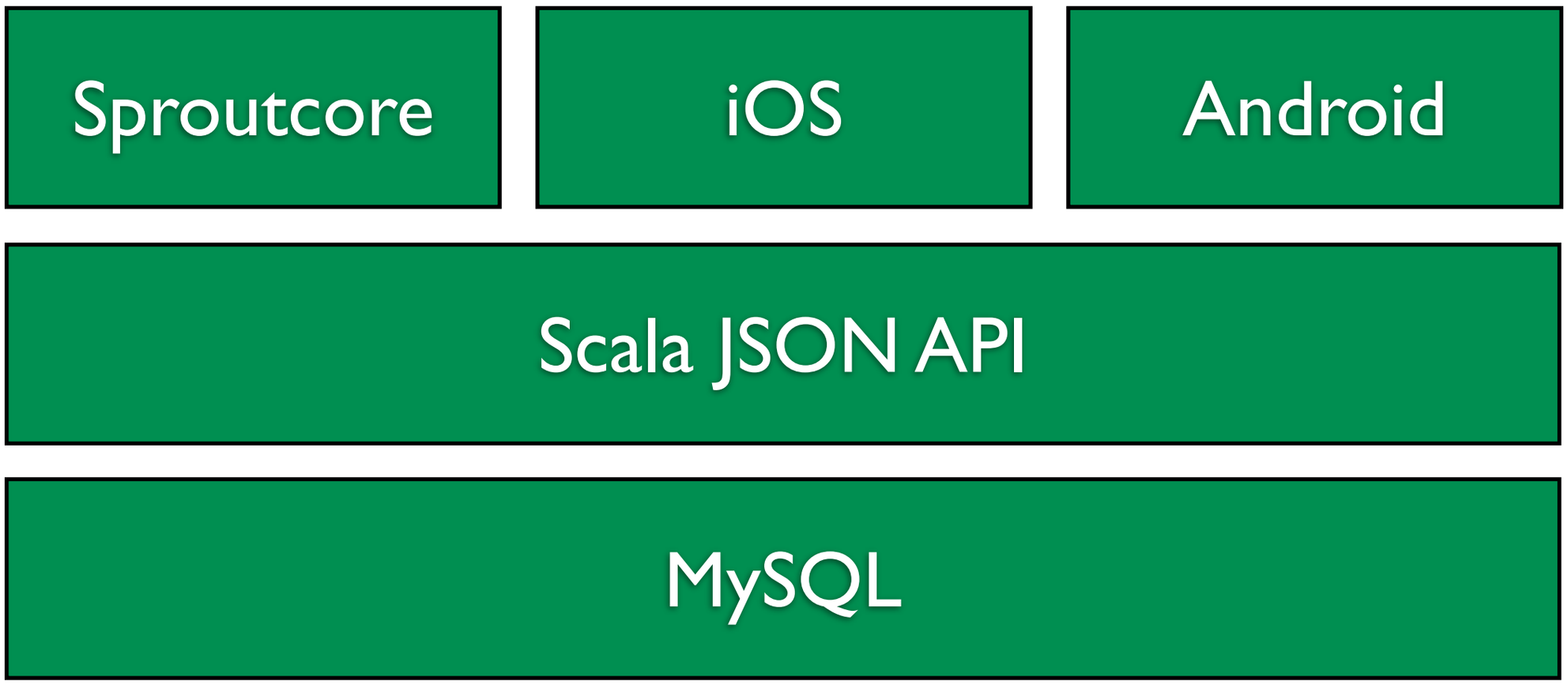
COMPOSED OF A
SCALA API SERVER

We use scala to build our JSON API server, it provides authenticated / authorized accesses to our datastore. Handles some notifications and other things. Slowly expanding to include other things.

AND...



A rich javascript client



STACK

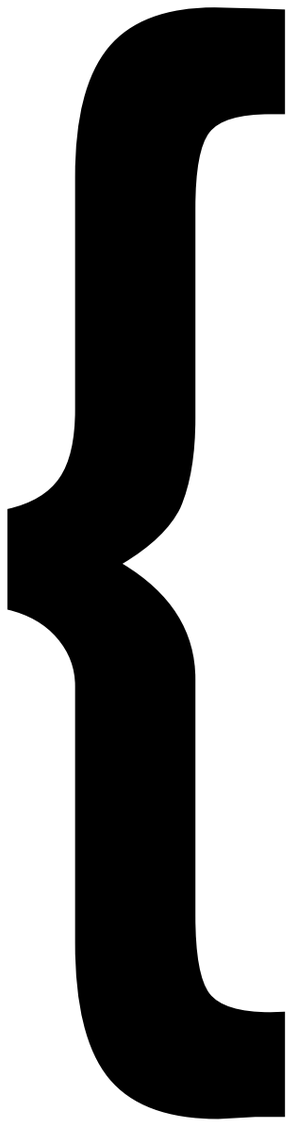
We also use that API to power our mobile clients – JSON everywhere!

Scala JSON API

STACK

We also use that API to power our mobile clients - JSON everywhere!

SCALA API

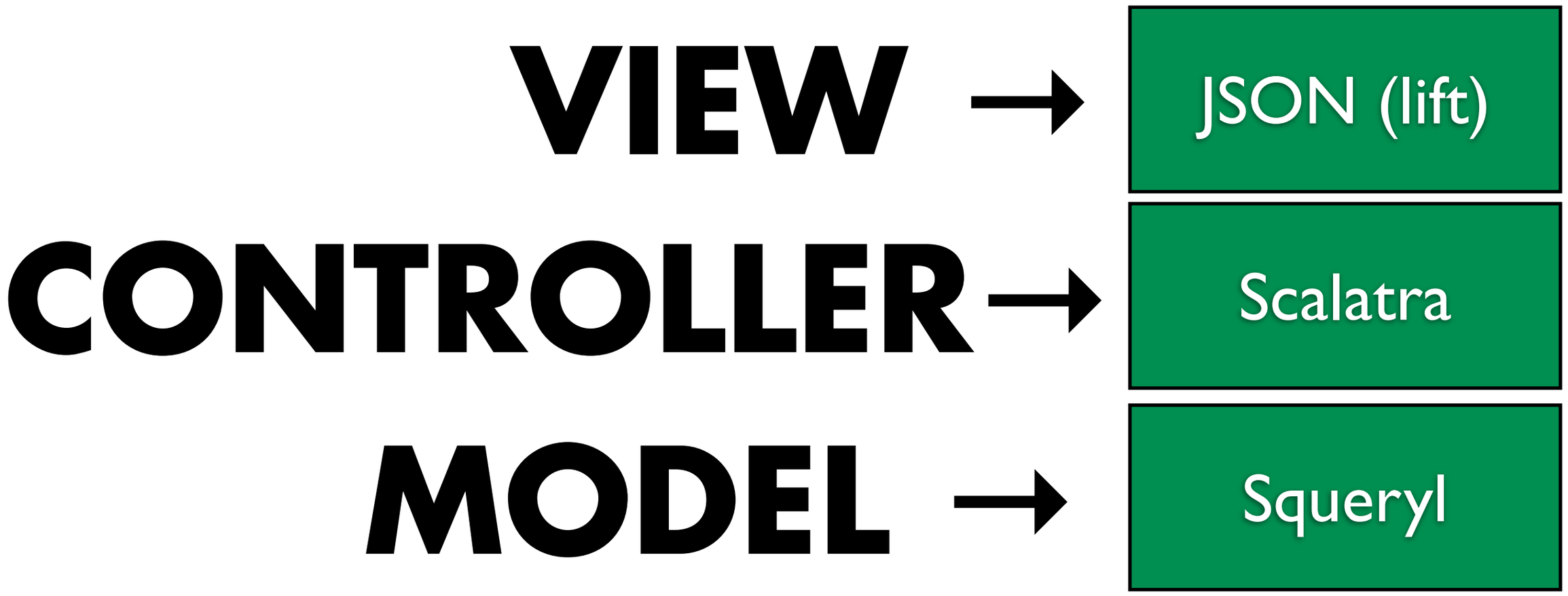


JSON (lift)

Scalatra

Squeryl

When we first made the switch to scala there was basically Lift, which we deemed to be a bit of a headache. Play 2.0 (Scala) was on the horizon so we ended up building our own little web stack on three libraries. Everyone likes MVC so you can really look at our back end like an MVC that is powered by these three libraries and some glue code written in Scala.



When we first made the switch to scala there was basically Lift, which we deemed to be a bit of a headache. Play 2.0 (Scala) was on the horizon so we ended up building our own little web stack on three libraries. Everyone likes MVC so you can really look at our back end like an MVC that is powered by these three libraries and some glue code written in Scala.

LIBRARIES

Friday, 4 May, 12

M - SQUERYL

HTTP://WWW.SQUERYL.ORG/

ULTRA THIN ORM

- A DSL for writing SQL in Scala *cough*
- Define Relationship / Constraints / Indexes etc.
- Uses posos
- No DB evolution :(

Squeryl lets you write SQL in a compiler checked DSL which is great for protecting against schema changes but it also has the short coming of not having any provision for database evolution, like migrations. Some options for this include using liquibase, or AR migrations. We chose to use ActiveRecord migrations because they are really clean and easy to read / understand.

A TRIVIAL EXAMPLE

```
class Unicorn(val name:String) extends MatygoModel {

    def sayHello = "neigh-a-corn"

}

val unicornsNamedJoe = from(MatygoSchema.unicorns) (unicorn =>
                       where(unicorn.name === "Joe")
                       select(unicorn))

// SELECT * FROM unicorns u WHERE u.name = “Joe”
```

LIFE CYCLE AND VALIDATIONS

```
class Unicorn(val name:String) extends MatygoModel {

  validates (
    lengthOfString("name", _.name, 20, 40),
    formatOfString("name", _.name, """\S""".r)
  )

  def beforeCreate = ()
  def afterCreate = ()

  def beforeUpdate = ()
  def afterUpdate = ()

  def beforeDestroy = ()
  def afterDestroy = ()
}
```


When we started with Squeryl there were no life cycle methods on objects (there appear to be now), which prevented us from having some sort of sane validations, as well as having application layer triggers. So we've basically decorated the Squeryl crud methods methods, and added before and after hooks.

C - SCALATRA

HTTP://WWW.SCALATRA.ORG/

SCALATRA

- Based off Sinatra.rb
- Dead Simple Controllers
- Keeps routing out of web.xml
- Controllers are servlet filters

Really a pleasure to use, simple routing. Great for building rest services, we reflectively load our controllers at bootstrap and attach them to a base servlet. The project has a lot of active development, and in my opinion for a simple JSON service, it's a really great option.

SCALATRA

```
class UnicornController extends MatygoController {
    get("/unicorns"){
       Unicorn.all
    }

    get("/unicorns/:id"){
       Unicorn.find(params("id"))
    }

    post("/unicorns"){
       Unicorn(req.jsonBody)
    }

    put("/unicorns/:id"){
       Unicorn.find(params("id")).fromJSON(req.jsonBody, currentUser)
    }

    delete("/unicorns/:id"){
       Unicorn.find(params("id")).destroy
    }
}
```

SCALATRA

WITH AUTHORIZATION

```
class UnicornController extends MatygoController with MatygoAuthorization {
    get("/unicorns"){
        rejectNotRole(Unicorn.all, Role.Admin)
    }

    get("/unicorns/:id"){
        rejectNotOwnerOrRole(Unicorn.find(params("id")), Role.Admin)
    }

    post("/unicorns"){
        rejectNotRole(Role.Instructor)
        Unicorn(req.jsonBody)
    }

    put("/unicorns/:id"){
        rejectNotOwnerOrRole(Unicorn.find(params("id")), Role.Admin).fromJSON(req.jsonBody, currentUser)
    }

    delete("/unicorns/:id"){
        rejectNotOwnerOrRole(Unicorn.find(params("id"))).destroy
    }
}
```

V - LIFT-JSON

HTTPS://GITHUB.COM/LIFT/LIFT/TREE/MASTER/FRAMEWORK/LIFT-BASE/LIFT-JSON/

LIFT-JSON

Implicit conversion from maps

```
val json = ("subject" -> subject) ~ ("content" -> content)
```

Obtuse extraction

```
val string = (json \ "subject").extract[String]

val stringOpt = (json \ "subject").extractOpt[String]
```


The strange extraction operators come from XPath,

LIFT-JSON

Serialization

```
def fromJSON(json: JObject) = {
    (json \ "subject").extractOpt[String].map(string => subject = string)
    (json \ "content").extractOpt[String].map(string => content = string)
    this
}

def toJSON = ("subject" -> subject) ~ ("content" -> content)
```


There are more advanced features in the LiftJSON package that allow for class / object extraction from JSON. We don't really make much use of those features, as we need more fine grained control over the serialization.

SPECS AND TESTS

SCALATEST + SCALTRA-SCALATEST

Earlier in the year, I refactored about 60% of our scala code in some way or another. Greatly simplifying the how we handle requests, and organize datasets. The scalatest framework was critical in the safety and efficacy of this refactoring.

We use full stack acceptance tests written as scalatest specs, and have somewhere in the high 90s coverage. This allowed us to guarantee that changes to the underlying architecture would not effect the deployed clients.

TOOLS

Friday, 4 May, 12

SBT

(NOT SO) SIMPLE BUILD TOOL

SBT is a pain. It's not simple, but it is really powerful. The people at TypeSafe are working on it, and making a lot of headway we are assured.

SBT

- .sbt vs .scala
  - pick one - mixed projects suck
- Jenkins Integration (CI)
- plugins
  - xsbt-web-plugin - jetty
  - sbteclipse-plugin - eclipse class path magic
  - sbt-assembly - deploying fat jars

Friday, 4 May, 12

To make things a bit more sane, we avoid mixed .sbt and .scala projects. .scala ones tend to be a bit more powerful, I am told if you have complex multiple build versions they are handy, so if you need some crazy stuff it's not a bad road to take. But in our experience it quickly gets out of hand, and tends to be really free form.

Jenkins Integration exists... but is a bit on the rough side as well. We make do by shelling out for the most part.

So we stick to .sbt (for now)

We use a collection of plugins to make our life a bit better, most importantly to simplify deploy we use sbt-assembly, which lets us deploy a fat jar - makes the build a bit slower, but really who cares about that stuff.

SBT-ASSEMBLY

```
excludedFiles in assembly := { (bases: Seq[File]) =>
  bases flatMap { base =>
    (base / "META-INF" * "*").get collect {
      case f if f.getName == "something" => f
      case f if f.getName.toLowerCase == "license" => f
      case f if f.getName.toLowerCase == "manifest.mf" => f
      case f if f.getName.toLowerCase.contains(".sf") => f
      case f if f.getName.toLowerCase.contains(".dsa") => f
      case f if f.getName.toLowerCase.contains(".rsa") => f
  }
}}
```


You need this if you are using any secured jars...

IDEs

ERRRRRR.... ECLIPSE

I don't like Intelli-j or Netbeans or Vim or Emacs.... But I know lots of people do, I cut my teeth mostly on Java projects, and for my money eclipse has the strongest refactoring tools - and all around my brain just works in eclipse.

Typesafe is putting most of their work into Eclipse right now, but as I understand they have plans for Intelli-J netbeans as well.

IT'S 2012...

MANAGE MY OWN IMPORTS
DO SEARCH AND REPLACE REFACTORING
FIND SYNTACTIC ERRORS AT COMPILE TIME
CODE FOLLOW USING ACK

IT'S 2012...

MANAGE MY OWN MEMORY

DO SEARCH AND REPLACE REFACTORING

FIND SYNTACTIC ERRORS AT COMPILE TIME

CODE FOLLOW TISING ACK

COMPLEXITY VS COMPILE TIME

- Uses SBT under the hood
- 14488 Lines of Scala code
- 30s clean compile
- 25s change to highly referenced file
- 4s change to minor file

This wasn't always the case though, when we got started on this eclipse-ide was prohibitively slow (ala 2 min compile times) and the compiler would randomly kick in - even when disabled. So we spent a lot of time working in TextMate/ vim / etc.

Typesafe has thrown a lot of money / effort into eclipse-ide and it's now quite a bit better to hang around in.

Additionally, coming down the pipe relatively soon, we are going to see some great improvements in scala tests, allowing you to do things like run individual specs (ala JUnit + Java)

GET A REAL COMPUTER...

- 4 GB of heap space allocated to sbt(8gb of physical ram)
- 2.2 gHz quad core proc (hyper threaded)
- SSD

THINGS I LIKE

Friday, 4 May, 12

PYTHON PARADOX

Friday, 4 May, 12

When Paul first suggested to me that we look at Scala, one of his arguments was that the Python Paradox may also apply to Scala. That is to say that given an random sampling of developers on average Scala developers may be sharper.

In my approximation the Scala community is roughly divided into Functional Programmers, Language enthusiast / academics, and the rest are Java Refugees.

So far this has proven to be mostly true, the calibre of people we talk to who are interested in us because we use Scala has been impressive, and in general the idea of getting to work with exciting new languages tends to attract the cream of the crop.

SCALA PARADOX

When Paul first suggested to me that we look at Scala, one of his arguments was that the Python Paradox may also apply to Scala. That is to say that given an random sampling of developers on average Scala developers may be sharper.

In my approximation the Scala community is roughly divided into Functional Programmers, Language enthusiast / academics, and the rest are Java Refugees.

So far this has proven to be mostly true, the calibre of people we talk to who are interested in us because we use Scala has been impressive, and in general the idea of getting to work with exciting new languages tends to attract the cream of the crop.

COMPOSITION

```
class Target extends A {

    val leftOperand = 5

}
```

```
abstract trait A {

    def leftOperand:Int

    def product(value: Int) = leftOperand * value

}

object Test {
    val target = new Target
    val product = target.product(4) // 20
}
```


Traits are fantastic, composition using the template pattern is gold. We use this pattern all over the place to encapsulate functionality in small block of reusable + testable code.

DECORATION

```
trait A {
    def speak = "A!"
}

trait B extends A {
    override def speak = "B(" + super.speak + ")"
}

trait C extends A {
    override def speak = "C(" + super.speak + ")"
}

class TestObject extends A with B with C {
    override def speak = "TestObject(" + super.speak + ")"
}

object Test extends App{
    println(new TestObject().speak)
    //TestObject(C(B(A!)))
}
```


I am also a big fan of creating decorated objects again using Scala's traits. This is one of the real powers of the mixin style pattern, that you can just bolt extra behaviour onto and object by adding a trait and as long as it's built with decoration in mind (calls it's super.someMethod) things play nice.

HIGHER ORDER FUNCTIONS

FOLDS / MAPS / FILTERS / MAGIC!

If you spend any sufficient amount of time in a more “modern” language you’ll like encounter / come to love, first class / higher order functions.

When we were still doing Java, we built out some classes to help us do things like folds & maps in Java with anonymous inner classes and generics. They had a lot of noise, but were better than nothing. Scala’s inclusion of things like this really makes life a lot better for trivial tasks on small sets of data.

THINGS I DON'T

Friday, 4 May, 12

OPERATOR MADNESS

```
val list = List(1, 2, 3, 4, 5, 6)

list.foldLeft(0)((a, b) => a+ b)

list./:(0)((a, b) => a + b)

(0 /: list)((a, b) => a + b)
```


First off, the fact that operators are aliased to functions, can mean that on a team of five people without an agreed standard you'll end up with a little of all three. Also the fact that methods don't need to have the . in order to be called again can lead to messes all over the place.

IDIOMATIC SCALA?

The promise of a mixed paradigm language is great, however at least for me, the fact that there are at least 5 different ways to express nearly every control structure, or execute any given method is frustrating.

Especially if you are new to the language and heading out looking for help.

So far as I can tell, idiomatic Scala has yet to emerge. That said, I think there is a lot of great work being done, and I can really see the future, and it looks promising.

OOP

WITH SOME FUNCTIONAL FLARE

At Matygo we essentially do Object Oriented programming with some functional bits where it feels good / right.

This decision has to do with essentially with a combination of my own competencies and experience, and a hypothesis that it will be easier to onboard new developers into our code base, when it is designed this way.

QUESTIONS?

WANT A JOB?

JOE@MATYGO.COM
PAUL@MATYGO.COM