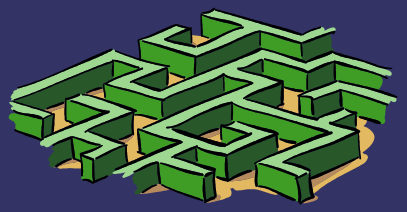
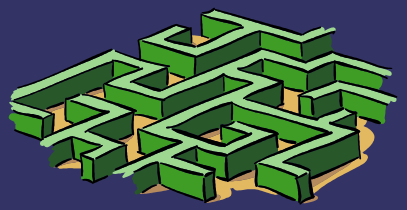
Play! 2 With Scala

Aaron Allred
@digicyc



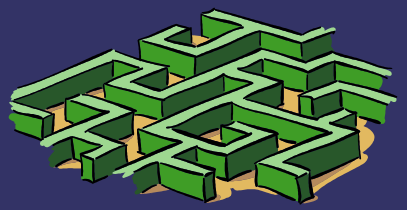
About Moi

- Insanely obsessed with Scala.



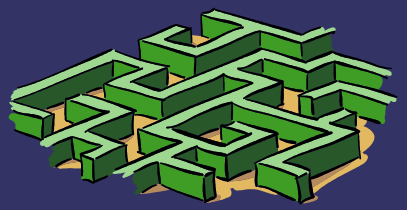
About Moi

- Insanely obsessed with Scala.
- Insanely bad at Presenting.



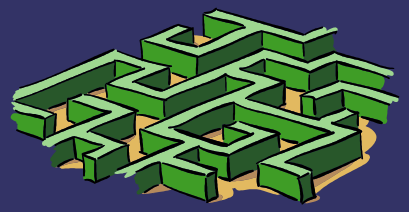
About Moi

- Insanely obsessed with Scala.
- Insanely bad at Presenting.
- Have sadly gotten to use Scala less lately.



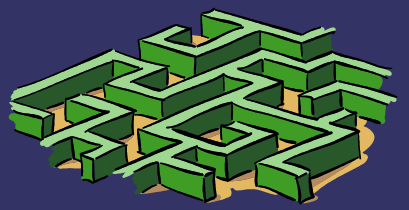
About Moi

- Insanely obsessed with Scala.
- Insanely bad at Presenting.
- Have sadly gotten to use Scala less lately.
- Was a hardcore Lift junky.



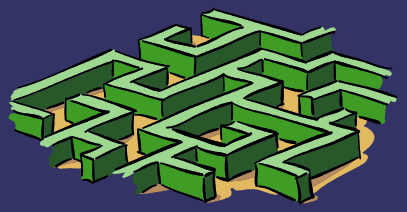
Lets get to PLAY

- Super easy to jump into and start coding.



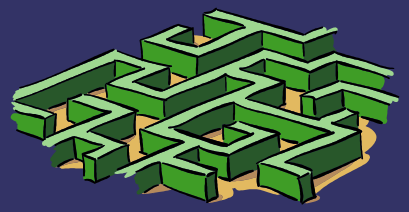
Lets get to PLAY

- Super easy to jump into and start coding.
- Utilizes the Scala Compiler to type check everything.



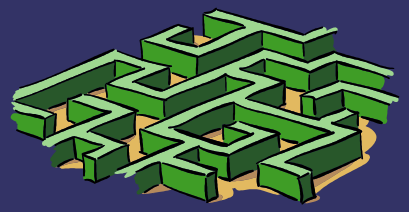
Lets get to PLAY

- Super easy to jump into and start coding.
- Utilizes the Scala Compiler to type check everything.
- Based on event-driven, non-Blocking IO.



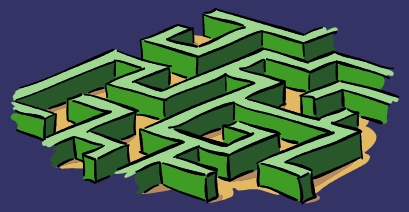
Lets get to PLAY

- Super easy to jump into and start coding.
- Utilizes the Scala Compiler to type check everything.
- Based on event-driven, non-Blocking IO.
- Based on a Stateless and well known MVC model.



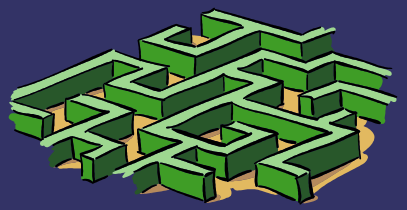
Lets get to PLAY

- Super easy to jump into and start coding.
- Utilizes the Scala Compiler to type check everything.
- Based on event-driven, non-Blocking IO.
- Based on a Stateless and well known MVC model.
- Easy to Scale.



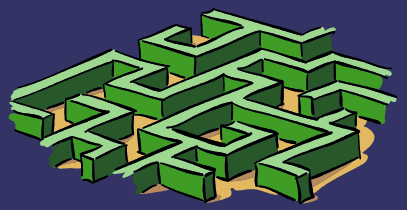
Lets Play!

How to Start using Play.



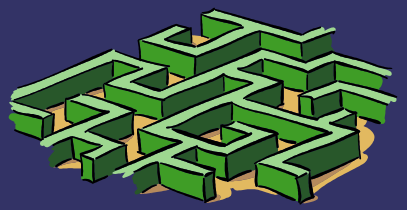
Getting Play!

- Download the ZIP file from http://playframework.org



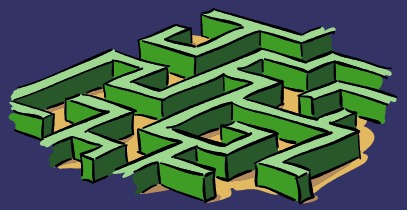
Using Play!

- Download the ZIP file from http://playframework.org
- Create new project with '**play new <appname>**'



Using Play!

- Download the ZIP file from http://playframework.org
- Create new project with '**play new <appname>'**
- **- cd <appname>;play run**
- Goto *http://localhost:9000*

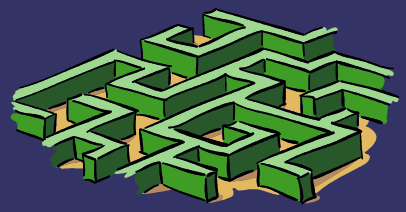


Coding Play with our Existing Tools

- Eclipse = '***play eclipse***'
  This will generate Eclipse Project files

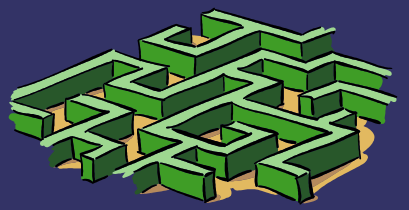
- IntelliJ = '***play idea***'
  This will generate IDEA Project files.

- Older Play versions have different commands.
  '***play eciplisify***'



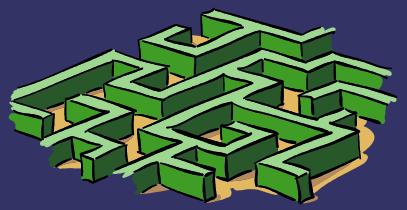
OMG MVC, we know this!

- M.odel
- V.iew
- C.ontroller



Controller/

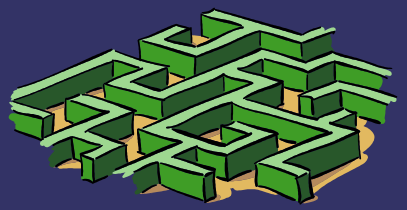
- Most requests received by a Play app are handled by an ***Action.***



Controller/

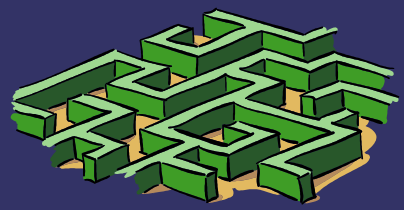
- Most requests received by a Play app are handled by an ***Action***
- An Action is

play.api.mvc.Request => play.api.mvc.Result



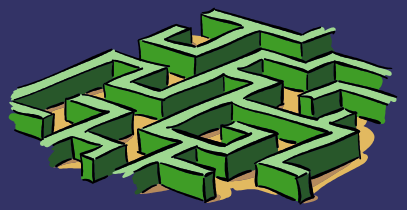
Common Action Results

```
1 val ok = Ok("Hello world!") // HTTP 200
2 val notFound = NotFound
3 val pageNotFound = NotFound(<h1>Page not found</h1>)
4 val badRequest =    BadRequest(views.html.form(formWithErrors))
5 val oops = InternalServerError("Oops")
6 val anyStatus = Status(488)("Strange response type")
```



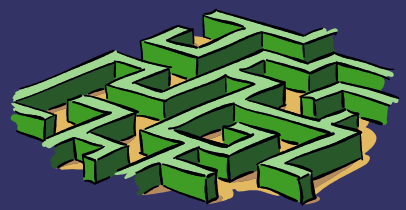
So what Controls the Controller?

- How do we connect URL's to our Functions?



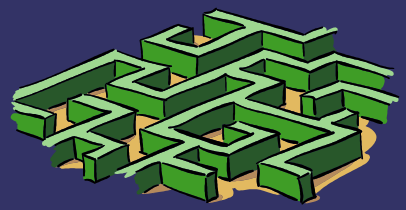
'conf/routes' *Connects URLs to Functions*

```
 1 # Routes
 2 # This file defines all application routes (Higher priority routes first)
 3 # ~~~~
 4
 5 # Home page
 6 GET     /                           controllers.Application.index
 7 POST    /addUser                    controllers.Application.addUser
 8
 9 # Map static resources from the /public folder to the /assets URL path
10 GET     /assets/*file               controllers.Assets.at(path="/public", file)
```

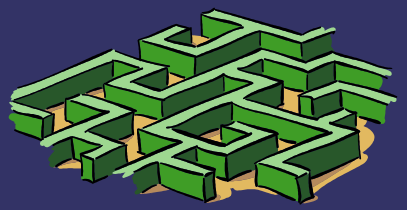


SUPER easy and mostly STATIC

```
object Application extends Controller {

 def index = Action {
  Ok(views.html.index("Home"))
 }

 def location = Action {
  Ok(views.html.location("Locations"))
 }

 def aboutus = Action {
  Ok(views.html.aboutus("About Us"))
 }

 def talks = Action {
  Ok(views.html.talks("Talks"))
 }
}
```



So now how do we Present our Sexy HTML Skillz?!



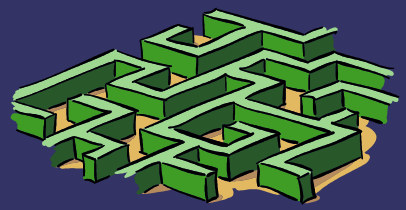
So now how do we Present our Sexy HTML Skillz?!

- In ***app/views*** We see files of this nature:
- - main.scala.html
- - index.scala.html

- The ***views.html.index*** translates to the file ***views/index.scala.html***

```
def index = Action {
 Ok(views.html.index("Home"))
}
```

"Home" is the Parameter passed to the Template.



So now how do we Present our Sexy HTML Skillz?!

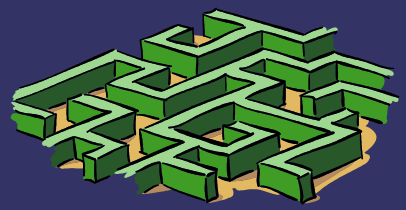
- In ***app/views*** We see files of this nature:
- - main.scala.html
- - index.scala.html

- The ***views.html.index*** translates to the file ***views/index.scala.html***

```
def index = Action {
 Ok(views.html.index("Home"))
}
```

"Home" is the Parameter passed to the Template.

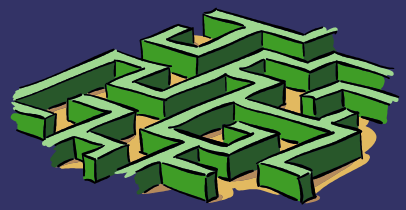
SAY WHAT?



OMG This looks like a Scala Function with XML. And main appears to have a Curried parameter List!

File: 'index.scala.html'

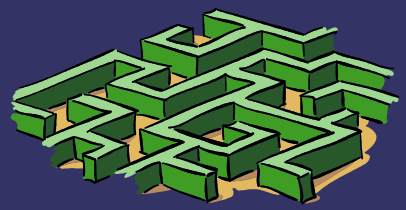
```
@(message: String)

@main(message) {

  <div class="hero-unit">
    <h2>Welcome to USEScala!</h2>
    <p>Home of the Utah Scala Enthusiasts</p>
    <p>
      <a class="btn btn-primary btn-large">
        Learn more &raquo;
      </a>
    </p>
  </div>
}
```



OMG This looks like a Scala Function with XML. And main appears to have a Curried parameter List!

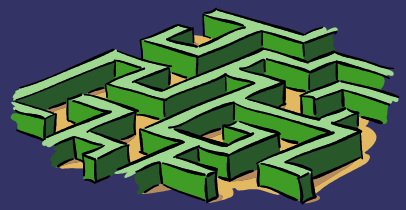
That's because it is and it does.

```
@(message: String)    // message is a function argument of type String
                      // This is how we passed “Home” to it.
@main(message) {      // main is a function with a Curried Param List. (String)(Html)

  <div class="hero-unit">
    <h2>Welcome to USEScala!</h2>
    <p>Home of the Utah Scala Enthusiasts</p>
    <p>
      <a class="btn btn-primary btn-large">
        Learn more &raquo;
      </a>
    </p>
  </div>
}
```



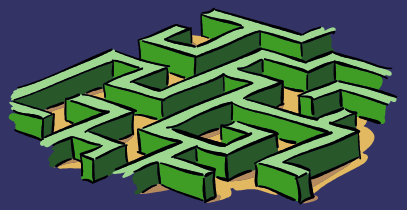
Freedom of control in Templates

```
@(products: List[Products])

<ul>
@products.map { p =>
 <li>@p.name ($@p.price) </li>
}
</ul>
```



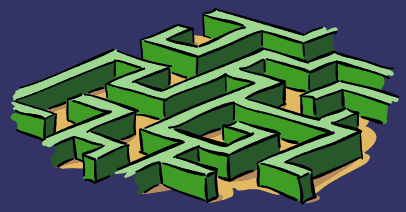
Models

- Has Anorm which really isn't an ORM
- You can use whatever you want. Will fit in easy.



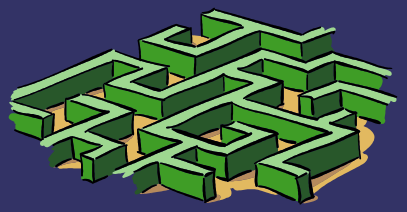
Models and Database support in Play.

- Some built in Database support in Play
- Evolution and Anorm
- Evo's exist in ***'conf/evolutions/default'***
- Naming scheme: ***1.sql 2.sql***



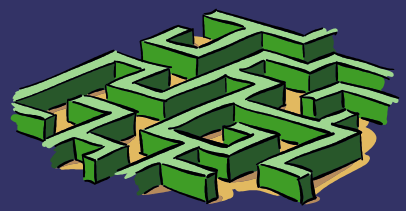
evolutions

- The **!Ups** describes the required transformations.
- The **!Downs** describes how to revert them.

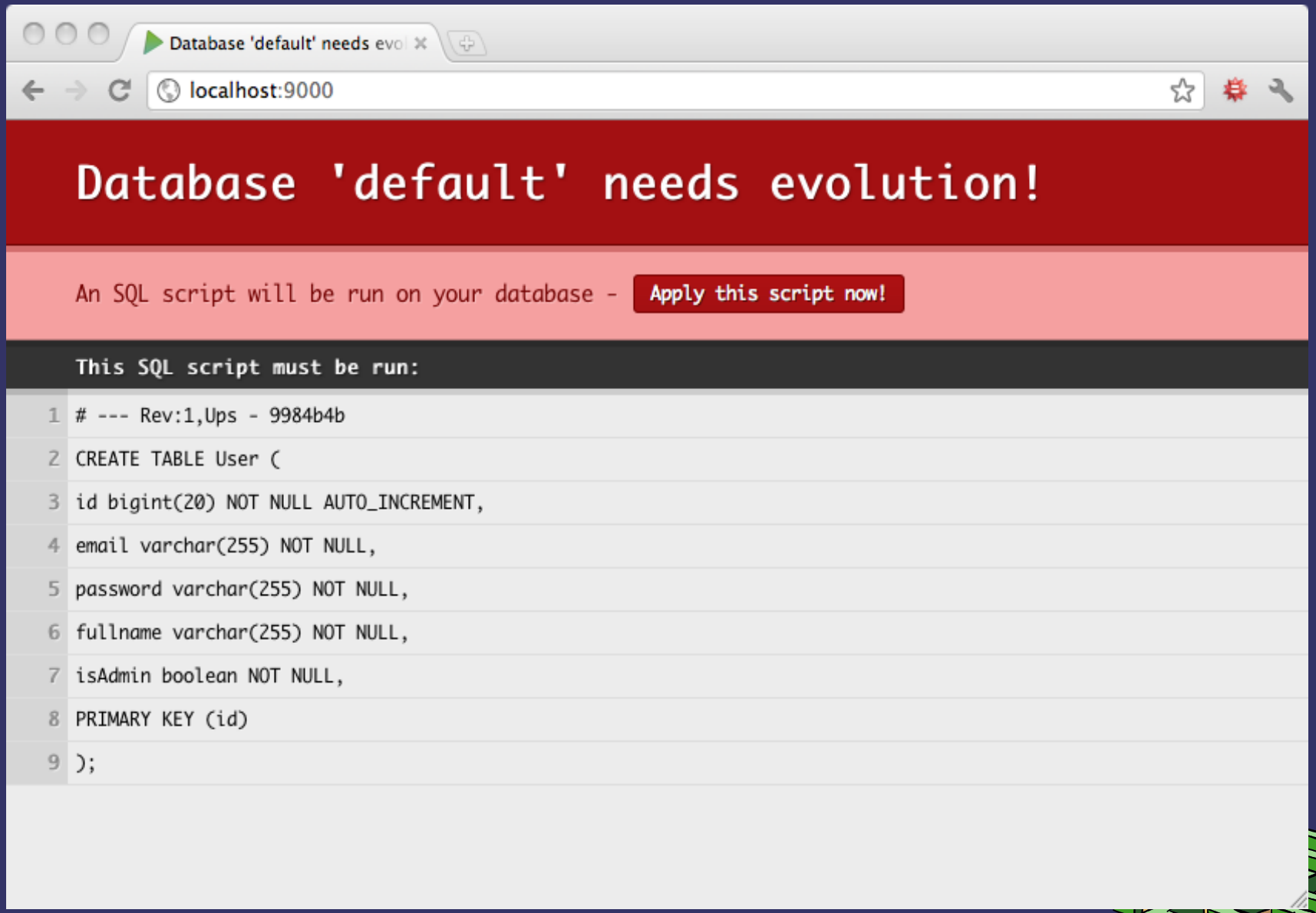


evolutions

```
1  # Users schema
2
3  # -- !Ups
4
5  CREATE TABLE User (
6    id bigint(20) NOT NULL AUTO_INCREMENT,
7    email varchar(255) NOT NULL,
8    password varchar(255) NOT NULL,
9    fullname varchar(255) NOT NULL,
10    isAdmin boolean NOT NULL,
11    PRIMARY KEY (id)
12 );
13
14 # -- !Downs
15 DROP TABLE User;
```

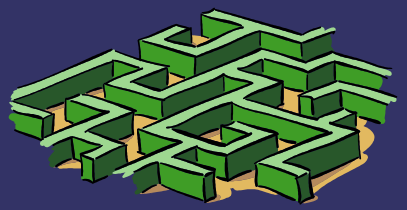


First time run through in Dev Mode



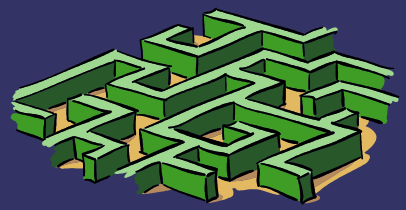
Anorm

- The power of Raw **SQL**
- Retrieve data using the **Stream API**



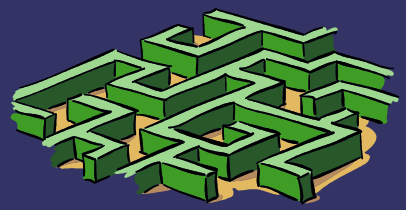
Using Pattern Matching

```
case class SmallCountry(name:String)
case class BigCountry(name:String)
case class France

val countries = SQL("Select name,population from Country")().collect
{
 case Row("France", _) => France()
 case Row(name:String, pop:Int) if(pop > 1000000) => BigCountry(name)
 case Row(name:String, _) => SmallCountry(name)
}
```



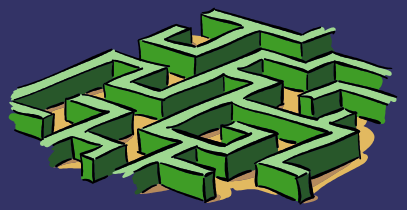
Raw SQL

```
val id: Int = SQL("insert into City(name, country) values ({name},
{country}")
       .on("Cambridge", "New Zealand").executeInsert()

val sqlQuery = SQL(
 """
  select * from Country c
  join CountryLanguage l on l.CountryCode = c.Code
  where c.code = 'FRA';
 """
)

SQL(
 """
  select * from Country c
  join CountryLanguage l on l.CountryCode = c.Code
  where c.code = {countryCode};
 """
).on("countryCode" -> "FRA")
```



BDD

- Writing Tests with our Favorite Test Frameworks..

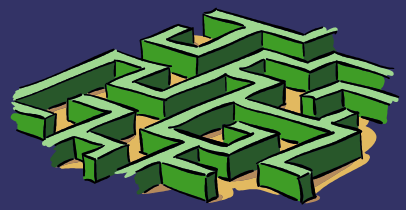


BDD

- Writing Tests with our Favorite Test Frameworks..
- Specs2

BDD

- Create directory in root of project: 'test/'

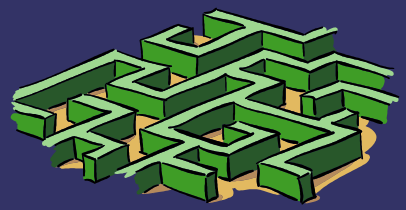


TestServer Spec

```
import org.specs2.mutable._

import play.api.test._
import play.api.test.Helpers._
import play.api.libs.ws.WS

class TestServer extends Specification {
 "The 'TestServer'" should {
  "run in a server" in {
   running(TestServer(3333)) {
    await(WS.url("http://localhost:3333").get).status must equalTo(OK)
   }
  }
 }
}
```

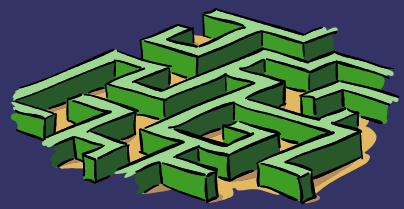
Super simple Controller Spec

```
import org.specs2.mutable._

import play.api.test._
import play.api.test.Helpers._
import play.api.mvc._

class TestController extends Specification {
 "My controller test" should {
  "respond to the index Action" in {
   val result = controllers.Application.index()(FakeRequest())

   status(result) must equalTo(OK)
   contentType(result) must beSome("text/html")
   charset(result) must beSome("utf-8")
   contentAsString(result) must contain("USEScala")
  }
 }
}
```



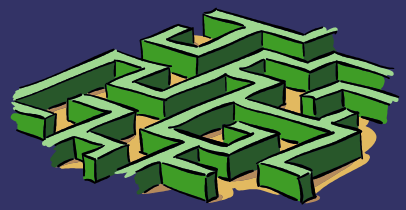
Simple Test Router Spec

```
import org.specs2.mutable._

import play.api.test._
import play.api.test.Helpers._
import play.api.mvc._

class TestRouter extends Specification {
 "My router test" should {
  "respond to the index Action" in {
   val Some(result) = routeAndCall(FakeRequest(GET, "/"))

   status(result) must equalTo(OK)
   contentType(result) must beSome("text/html")
   charset(result) must beSome("utf-8")
   contentAsString(result) must contain("USEScala")
  }
 }
}
```



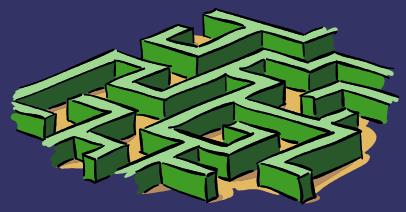
Simple Template Test Spec

```
import org.specs2.mutable._

import play.api.test._
import play.api.test.Helpers._
import play.api.mvc._

class TestTemplate extends Specification {
 "My template test" should {
  "render index template" in {
   val html = views.html.index("USEScala")

   contentType(html) must equalTo("text/html")
   contentAsString(html) must contain("USEScala")
  }
 }
}
```



So much more I'm not ready to cover

- Code LESSCSS and play auto compiles and puts into project.
- Code CoffeeScript and have it auto add to project.
- Use Google Closure Compiler in the same sense.
- Easy deployment to Heroku with detail instructions on how.



Credits

- #scala on Freenode
- Http://www.playframework.org
- USEScala for introducing me to Play