Overview
Orientation

Orientation

Anatomy

Demonstration

Language Flexibility

Developer Friendly

Predictable Scaling

Modern Web Considerations

Overview Orientation Anatomy Demo Conclusion
What is Play?

- Web application framework
  - Standard MVC design pattern
- Designed with a “lightweight, stateless, web-friendly architecture”
  - Atypical for Java-based frameworks
- Written in Scala
  - Developers can write in either Scala or Java
- Compiled code runs on the JVM
  - JVM can certainly be fast enough (Twitter adopted in 2011)
- Opinionated
  - Convention over configuration
Who makes Play?

- Typesafe Inc.
  - Founded by the creators of Scala and Akka
- Zengularity
  - French web development studio
- Open Source (Apache 2 license)
  - 358 contributors on GitHub
- You?
Core Features

- Typesafe Activator for quick project creation and templating
- SBT dependency management with Maven Central as a repository
- Integrated JUnit and Selenium for testing
- Scala template engine (can be used for HTML)
- Asset compiler for CoffeeScript and LESS
- Multi-environment awareness

Built-in support for
- Websockets, Comet, EventSource
- RESTful HTTP interactions
- JSON/XML parsing and marshalling via Jackson
- Validation
- Authentication
- JPA
- SMTP
- Scheduling
Differences

- Statelessness
  - JavaEE normally operates using states

- Static methods
  - Controllers and methods are by default static as opposed to container-managed instances

- Asynchronous I/O
  - Utilizes Servlet 3.0 (JSR 315) to avoid using HTTP threads to execute business logic

- Hot-reloading
  - Saves wasteful cycles of application startup

- Modularity
  - Play modules inspired by Ruby modules and Django packages
Maturation

- First release in 2007
- 1.0 release in October 2009
- 2.0 release in March 2012
- Currently 2.3.4
Let’s Talk I/O

First Rule of I/O: It’s Expensive, really expensive.

Fetching the Frontispiece to ‘The History of the Royal-Society of London’
By Wenceslaus Hollar from Wikipedia.org.
A Small History Lesson

• In the beginning there was CGI: Common Gateway Interface
  • The Good: Easy interface between applications and web servers.
  • The Bad: CGI spins a new application process up for every connection.

• Then there was FastCGI: Fast Common Gateway Interface
  • The Good: Runs several persistent processes.
  • The Bad: We’re still running a bunch of processes.
The Problem with Processes

If your application involves...
• Reading or Writing to Disk
• Reading or Writing to a Network Socket

You still have long lived processes eating up CPU time.
Lots of connections means lots of processes.
Non-Blocking, Async I/O

• Tons of ways to do it, there’s no silver bullet.
• Allows CPU to keep working, while I/O does its job.
• Why hasn’t this been done before?
  • Concurrency is hard
Why is it being done now?

The web demands it!

- Streaming Video
- Chat Application
- Large File Transfer
- Push Notifications
- Twitter streams...
Websockets

Continuous full-duplex connection stream, and it’s easy.
Who uses Play?

- LinkedIn
- The Guardian
- Coursera
- Klout
- Gawker Media
- Listd
- SmartAsset
- Mashape
- Valraiso
- Woisio
- Patch
- LucidChart
- Ocado
- Teech.io
- StreetLend
- Kifi
- Quantio
- konciergeMD
- Miavia
- You get the idea...

Orientation  Overview  Anatomy  Demo  Conclusion
Case Study: LinkedIn

- Rapid Iteration
- Java and Scala
- Reactive
- Open
- Supported
- Flexible
Anatomy

Orientation
Anatomy
Demonstration

Language Flexibility
Developer Friendly
Predictable Scaling
Modern Web Considerations

Overview  Orientation  Anatomy  Demo  Conclusion
Activator

- A “browser-based or command-line tool that helps developers get started”
- Browser Tool
  - Allows editing, debugging, running, testing, and compilation of Play applications
- Command-Line Tool
  - Helps create applications from templates
  - Runs, cleans, and tests applications and calls underlying SBT commands as necessary
Structure

Overview
Orientation
Anatomy
Demo
Conclusion
public static Result index() {
    return ok("Hello world!");
}
Views

Scala templates are used to render HTML.

```scala
@(movieFan: Fan, movies: List[Movie])

<h1>Welcome @movieFan.name</h1>

<ul>
  @for(movie <- movies) {
    <li>
      @(movie.getTitle() + "-" + movie.getYear())
    </li>
  }
</ul>
```
```scala
import play.PlayJava

name := """play-java"""

version := "1.0-SNAPSHOT"

lazy val root = (project in file(".")).enablePlugins(PlayJava)

scalaVersion := "2.11.1"

libraryDependencies += Seq(
  javaJdbc,
  javaEbean,
  cache,
  javaWs,
  "uk.co.panaxiom" %% "play-jongo" % "0.7.1-jongo1.0",
  "com.wordnik" %% "swagger-play2" % "1.3.10"
)
```

**build.sbt**

Defines all necessary dependencies.
A flat file defines all application routes. These are mapped to static methods on controllers. If controllers are managed instead, `@controllers` is used to seek an instance of a controller.
Promises

The basic flow of a promise is shown above.
Promises

Play implements and uses promises in Java like this.

```
Promise<Double> promiseOfPIValue = computePIAsynchronously();
Promise<Result> promiseOfResult = promiseOfPIValue.map(
    new Function<Double, Result>() {
        public Result apply(Double pi) {
            return ok("PI value computed: " + pi);
        }
    }
);
```
Promises

Or equivalently, in Java 8 using lambda expressions...

```java
Promise<Double> promiseOfPIValue = computePIAsynchronously();
Promise<Result> promiseOfResult = promiseOfPIValue.map(pi ->
    ok("PI value computed: " + pi)
);
```
What is a module?

- A module is just another Play application, except everything is optional
- Can contain plain Java code packaged as a JAR file
- Automatically loaded in the application
- Open source community
- Play has a module repository to aid the developer in finding the correct module
Examples

- Guice
  - Dependency injection within Play applications
- JSON minification
  - Allows JSON asset pretty-printing and minification
- MongoDB (Jongo, Morphia, Jackson)
  - Provide managed MongoDB access
- Play! Authenticate
  - A highly customizable authentication module for Play
Demonstration

Orientation

Anatomy

Demonstration

Language Flexibility
Developer Friendly
Predictable Scaling
Modern Web Considerations

Overview ➔ Orientation ➔ Anatomy ➔ Demo ➔ Conclusion
Takeaways

- Play is a high-productivity Java and Scala web application framework
  - Based on a lightweight, stateless, web-friendly architecture
  - Predictable and minimal resource consumption to promote scalability
  - Asynchronous I/O
  - Integrated JUnit testing
  - Developer friendly RESTful implementation
  - Built-in tools and functionality

- Why we chose Play:
  - Ease of Learning
  - High Speed of Development
  - Growing and dynamic community
  - Scalability
  - Frequent and integrated testing