# Workshop on Play and Akka (Verizon VDSI)

Nirmalya Sengupta

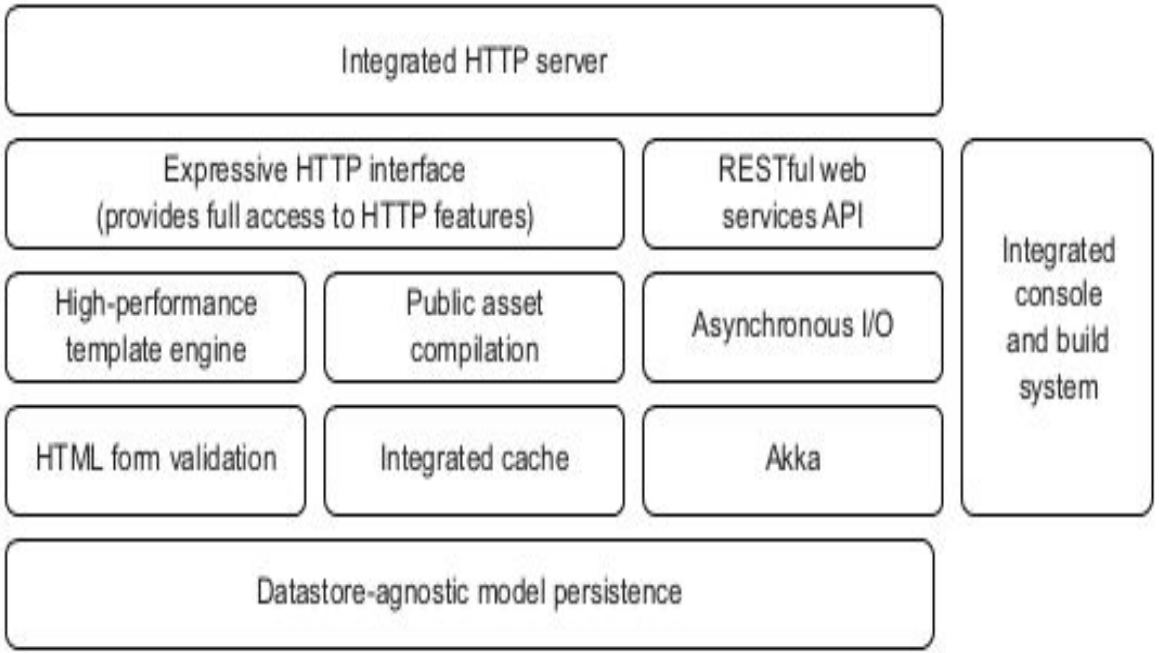
# Play days….

- A Web Framework written in Scala
- Has Scala and Java APIs
- Inspired by RoR and DJango
- Focus is on Web programming and ..
  - Not on Java EE (viz., Servlets)
- HTTP verbs are first class citizens

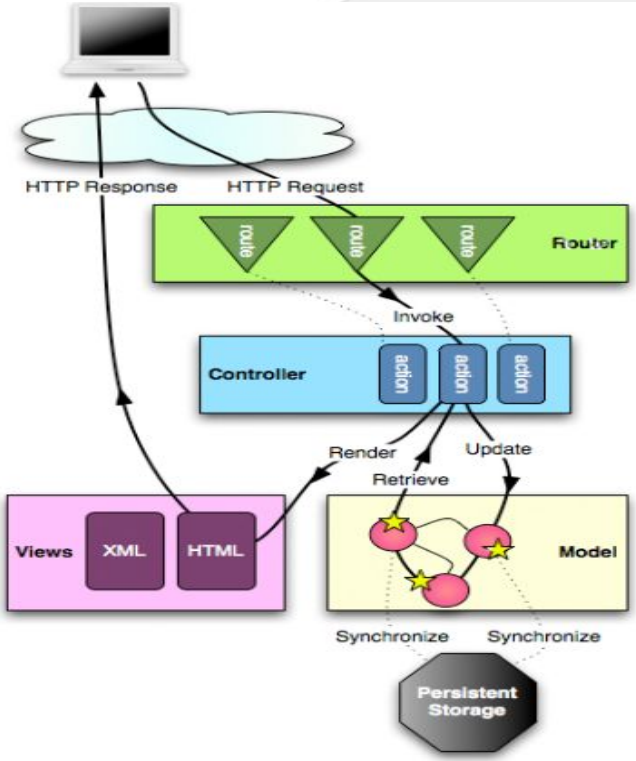
# Modern Web Applications require to be..

- Event Driven
- Scalable
- Responsive
- Resilient

# Play’s component stack..



# Play plays MVC ...




With thanks from: www.knoldus.com

# Play framework: your project directories

```
nirmalya@Cheetah:my-attendees-app-1$ tree -L 2 -d
.
├── app
│   ├── controllers
│   ├── filters
│   ├── models
│   ├── services
│   └── views
├── assets
├── bin
│   └── views
├── binary-guice-application
│   ├── bin
│   ├── libexec
│   ├── project
│   └── src
├── conf
├── libexec
├── logs
├── project
│   ├── project
│   └── target
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
```

# Play framework: M, V and C

- https://github.com/nsengupta/Play-Training-Elem-1
- Separation and distinct presence of:
  - Controller
  - View
  - Model
- Several types of Result available
  - ok, bad request, forbidden, payment required etc.
- **routes** is generated as a Class
- Controller is instantiated and injected in routes
- Implement action: **getBySurname()**

# Play framework: Exercise

- Modify my-attendees-app2 so that:
  - For every player, we also store the country he belongs to
- Add a new POST action to the controller, that
  - allows us to store the player's country as well
- Add a new GET action to the controller, that
  - returns the number of players from a given country
- https://github.com/nsengupta/Play-Training-Elem-2

# Play framework: Dependency Injection

- Supports JSR 330 (DI)
- Implements Guice Framework
  - Uses a Guice Application Loader
- Router is injected with Controllers (default)
- Components are created afresh, whenever needed and then injected (Singleton is explicitly defined)
- @Singleton directive

# Play framework: Dependency Injection

- Interface to Implementation binding
  - @ImplementedBy
  - Using modules
- Component Lifecycle
  - Created afresh, whenever needed
  - Lazily created (don’t create, if not used)
  - Starting/Stopping: Application LifeCycle

# Play framework: Posting new data

- Define the Form
- Define new Action(s)
- Form objects are immutable
  - A fresh form object is created with default values
- https://github.com/nsengupta/play-training-lab-3
- Complete Form-based posting for Star Soccer Players
- Implement a similar Service for Star Cricket Players
  - with an extra ‘age’ field

# Play framework: Application/component lifecycle

- From Play 2.5.x onwrds
- StartUp:
  - <module>.bind() is used for component-specific startup logic
- Shutdown:
- Use ApplicationLifecycle (automatically available)

```
class Hello @Inject() (ApplicationLifecycle: lifecycle)
extends Hello {

   lifecycle.addStopHook { () =>
Future.successful(connection.stop()) } }
```

# Play framework: Using Actors in Play

- ActorSystem is built in - available for injection
- All Actor APIs are available
- Remember: Fire-and-Forget
  - Who to **tell** to?
- https://github.com/nsengupta/Play-Training-Elem-5

# Play framework: Database access

- Configuration: default
- H2 - in-memory DB, helps in testing
- JDBC Driver, hence portable
- Available injectable DB instance
- Connection pool: careful
- https://github.com/nsengupta/Play-Training-Elem-6

# Exercise: PlayerRankProvider Service

- Create two databases: Cricket and Soccer Players
  a. Each Player has a ranking (integer)
- Provide access to Ranking of Cricket and Soccer players, by using
  a. In-memory hash-map (*attendeesDB*)
  b. H2 Database
- Provide a REST Endpoint to modify the ranking of a player