Scala & Play

Hands-On

CERN Spring Campus Spain

April 14th – 16th, 2014

Jan Janke (CERN)

Agenda

- Build a simple web app with Play in 5 steps
  1. Project setup
  2. Web application skeleton
  3. Persistence support
  4. Display some server fetched data
  5. Add a simple form

Before Starting …

- Prerequisite
  - Previous "Scala Theory" talk
  - Working installations of
    - JDK 7
    - Scala 2.10.4
    - sbt 0.13
    - MongoDB 2.4.9
    - git 1.8

Install according to instructions.
Add binaries to your path environment variable.

Get the sources (https://bitbucket.org/montchanais/springcampus2014/):
git clone https://montchanais@bitbucket.org/montchanais/springcampus2014.git

Verify Required Dependencies

- One last check:

```
macjja:~ jjanke$ java -version
java version "1.7.0_40"
Java(TM) SE Runtime Environment (build 1.7.0_40-b43)
Java HotSpot(TM) 64-Bit Server VM (build 24.0-b56, mixed mode)
macjja:~ jjanke$
macjja:~ jjanke$ scala -version
Scala code runner version 2.10.4 -- Copyright 2002-2013, LAMP/EPFL
macjja:~ jjanke$
macjja:~ jjanke$ mongo -version
MongoDB shell version: 2.4.9
macjja:~ jjanke$
macjja:~ jjanke$ git --version
git version 1.8.5.2 (Apple Git-48)
macjja:~ jjanke$
macjja:~ jjanke$ sbt --version
Loading /Users/jjanke/Dev/lib/sbt-0.13/bin/sbt-launch-lib.bash
sbt launcher version 0.13.0
```

Agenda

- Build a simple web app with Play in 5 steps
  1. Project setup
  2. Web application skeleton
  3. Persistence support
  4. Display some server fetched data
  5. Add a simple form

git checkout -f step-01

Step 01 – Basic Project Setup

- Create a new empty working directory
  - Reference it by *$SPRINGCAMPUS* env. variable

Content of the $SPRINGCAMPUS directory:

```
.
├── .gitignore
├── README.md
├── build.sbt
├── core
│   └── build.sbt
├── project
│   └── plugins.sbt
└── ui
    └── build.sbt
```

SBT Project Setup

- Separate business logic from UI
  - And also from web services and other interfaces

```
.
├── .gitignore
├── README.md
├── build.sbt
├── core
│   └── build.sbt
├── project
│   └── plugins.sbt
└── ui
    └── build.sbt
```

3 projects have been created
- root (main container project)
- core (business logic)
- ui (play web application)

Each project has its own build.sbt file.

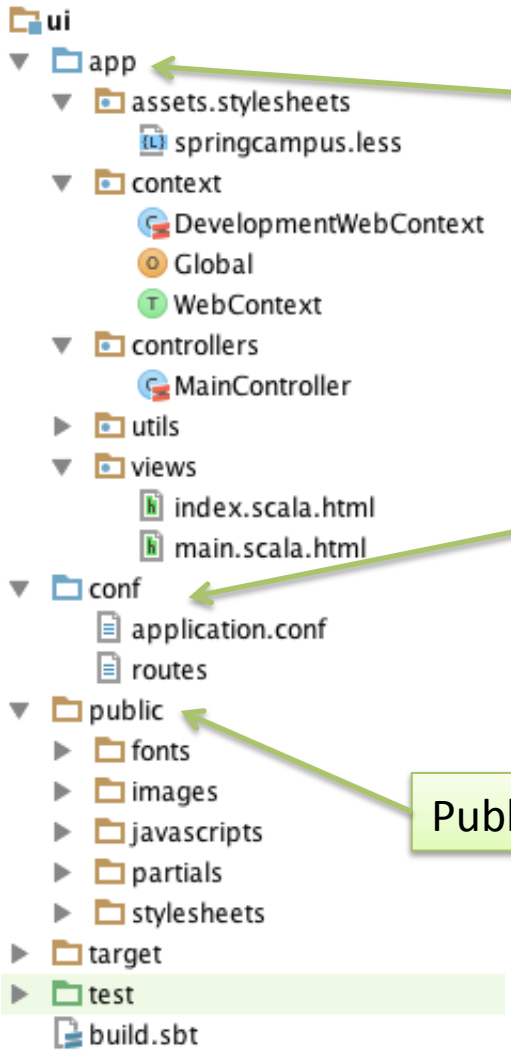
2 plugins are added to
- Support Play! web applications and
- Add support for IntelliJ IDEA project creation

Agenda

- Build a simple web app with Play in 5 steps
  1. Project setup
  2. Web application skeleton
  3. Persistence support
  4. Display some server fetched data
  5. Add a simple form

git checkout -f step-02

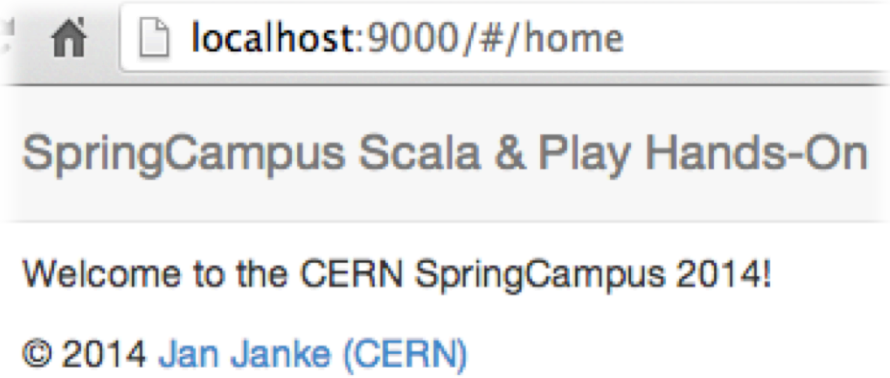
Step 02 – Simple Web Application



Play application artefacts incl. compiled LESS and Coffeescript files (domain models are not used in this application)

Configuration files

Public assets



Web Application Key Facts

- Use of third-party libraries
  - AngularJS and Bootstrap
- Play compiled Scala based templates
  - Use '@' to access any Scala object in scope
- Basic dependency injection (w/o Spring)
  - Look at Context, WebContext, Global
- In SBT make 'ui' project depend on 'core'

```
lazy val ui = project.in( file( "ui" ) )
  .dependsOn( core % "compile->compile;test->test" )
```

Dependency Injection with Play

- In conf/application.conf
  - application.global=context.Global
    - Global object provides hooks for
      - Application startup and shutdown, configuration etc.
      - Error handling
      - Controls controller creation
- In conf/routes
  - GET / @controllers.MainController.index

The '@' delegates controller retrieval to subclass of GlobalSettings in the method:

```
def getControllerInstance[A]( controllerClass: Class[A] )
```

Run the Play Application

Switch to 'ui' project

Launch sbt (from within the project directory)

```
macjja:springcampus2014 jjanke$ sbt
Loading /Users/jjanke/Dev/lib/sbt-0.13/bin/sbt-launch-lib.bash
[info] Loading project definition from /Users/jjanke/Dev/workspaces/vhx13/springcampus2014/project
[info] Set current project to root (in build file:/Users/jjanke/Dev/workspaces/vhx13/springcampus2014/)
> project ui
[info] Set current project to ui (in build file:/Users/jjanke/Dev/workspaces/vhx13/springcampus2014/)
[ui] $ ~run
```

Run the application (tilde makes it reload changed classes/resources)

```
--- (Running the application from SBT, auto-reloading is enabled) ---

[info] play - Listening for HTTP on /0:0:0:0:0:0:0:0:9000

(Server started, use Ctrl+D to stop and go back to the console...)

[success] Compiled in 551ms
[info] application - Starting SpringCampus web application...
[info] play - Application started (Dev)
```

Agenda

- Build a simple web app with Play in 5 steps
  1. Project setup
  2. Web application skeleton
  3. Persistence support
  4. Display some server fetched data
  5. Add a simple form

git checkout -f step-03

Step 03 – Persistence Support

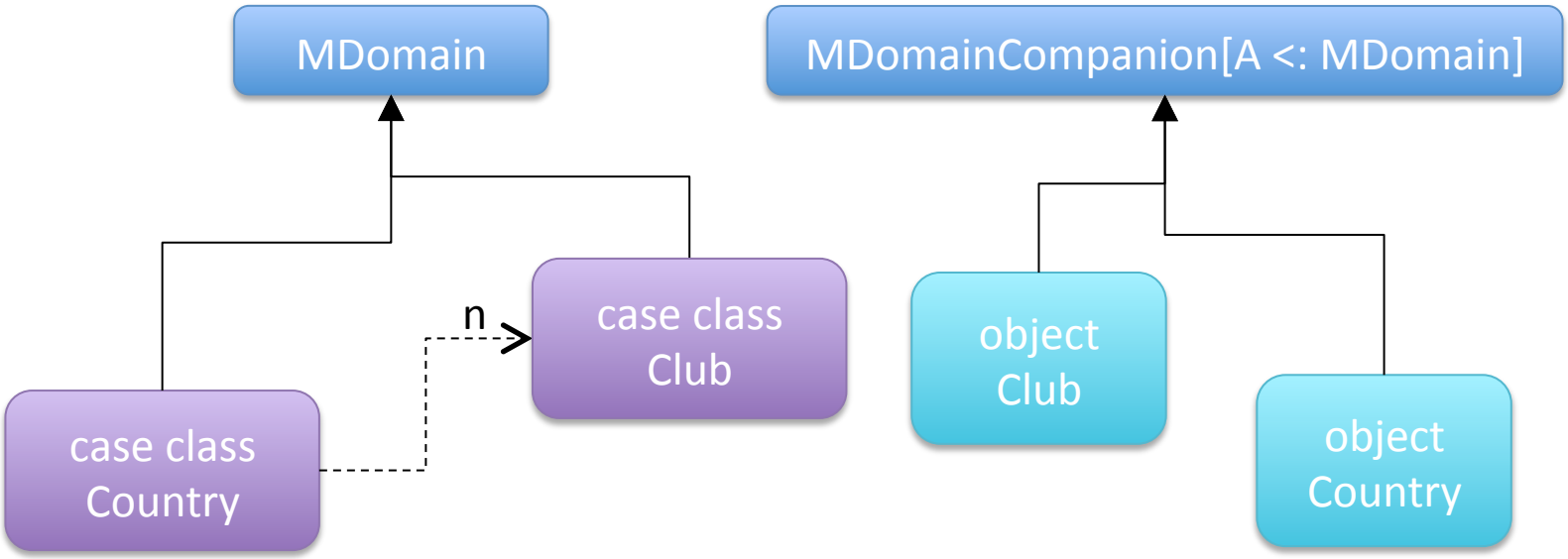
- springcampus.core
  - context
  - db
    - DomainUtils
    - MDomain
    - MDomainCompanion
    - MDomainCompanionRegistry
    - MEmbeddedDomain
    - MId.scala
    - MongoElement.scala
    - MongoPersistenceProvider
    - MongoQuery.scala
    - MRef.scala
    - PersistenceException.scala
    - PersistenceProvider.scala
  - domain
    - Club.scala
    - Country.scala
  - misc
  - service
    - InitializationService

Classes for MongoDB support

Domain model classes

Initialize the application / bootstrap test data

Domain Model



Domain instances represent the application data.

Domain companion objects provide service methods to create new MDomain instances.

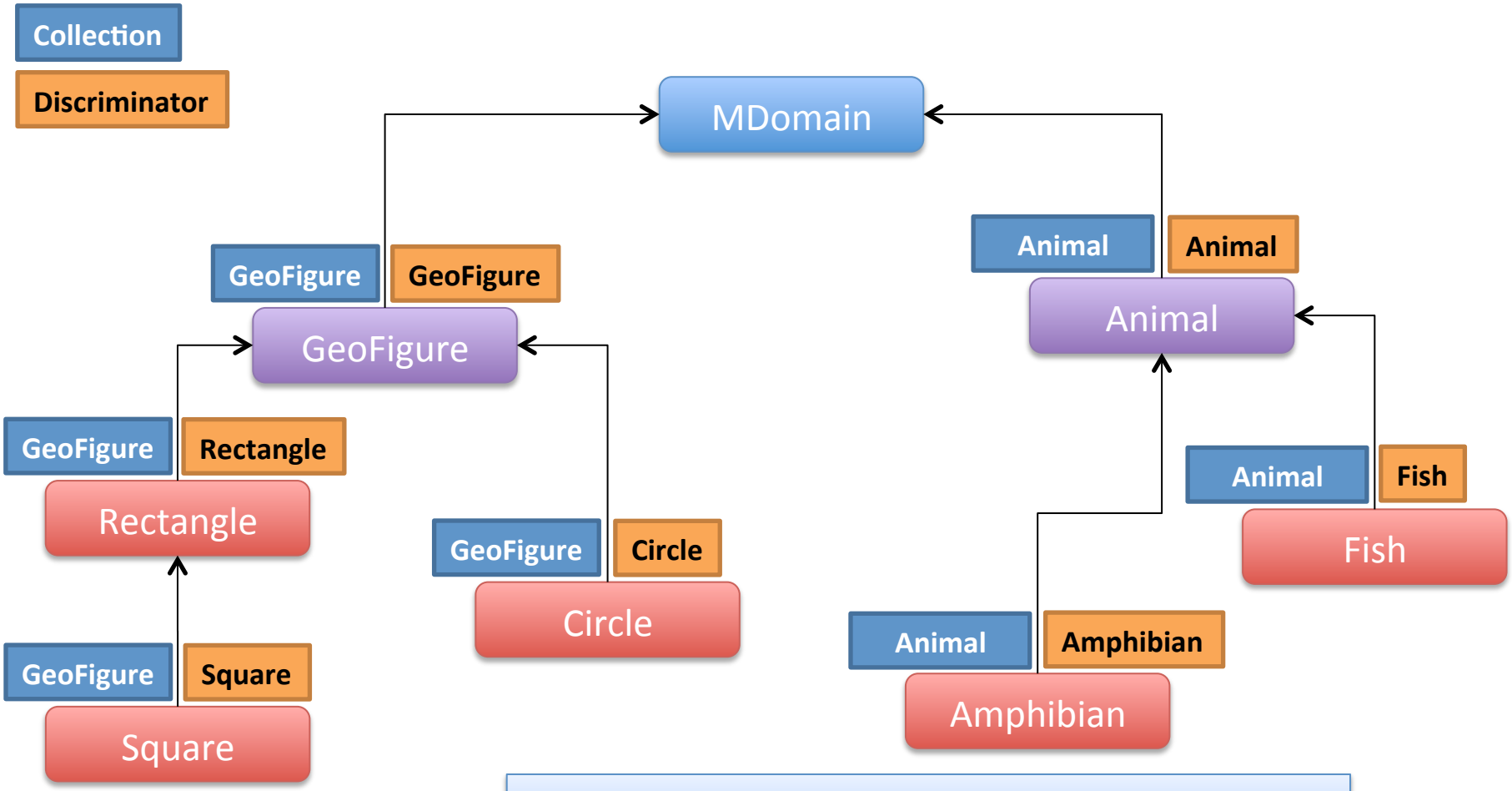
MongoDB Concepts

- Document store
  - Schemaless
  - Uses BSON format
    - Binary JSON with additional data types
- Data organized in collections
  - Collection is roughly equivalent to a table
  - In demo application
    - One collection per class hierarchy below Mdomain
    - Custom interface persists to / retrieves from MongoDB

A Typical Document

- Identifier
  - Unique within a collection (String, Long, ObjectID)
- Collection
  - Identifies the base class below MDomain
- Discriminator
  - Identified the actual class
- References
  - References other instances in the same DB

Collection vs. Discriminator



Collection identifies top-level class of a hierarchy.

Discriminator identifies the actual implementation class.

New Scala Concepts

- Sealed classes
  - Sub classes must be in the same source file
  - Allows for advanced compiler checks
- Variables and methods in one namespace
  - Functions can override variables and vice versa
- Scala reflection
  - Type tags (implicit type arguments)

(Simple) Reflection In Use

Definition of a type argument with a **type tag**

```
def findById[A <: MDomain : ru.TypeTag]( id: MId ): Option[A] = {

  val opResult = timedOperation[Option[A]]( logger.underlying.isTraceEnabled ){

    val coll = db.getCollection( DomainUtils.collectionName( ru.typeOf[A] ) )
    val cursor = coll.find( new BasicDBObject( MongoObject.ID.name, id.toBSON ) )

    withResource( cursor ) {
      if ( cursor.hasNext )
        Some( instantiateDomain( MongoObject.fromBSON( cursor.next() ) ) )
      else
        None
    }

  }

  logger.trace( "operation=findById; time={}ms; id={}; result={};",
    opResult.duration.toString, id, opResult.result.toString )

  opResult.result
}
```

Use tag to get type of type argument

Potential for optimization: Improve performance by using **macros**
(If log level 'trace' is not enabled, the compiler removes method call)

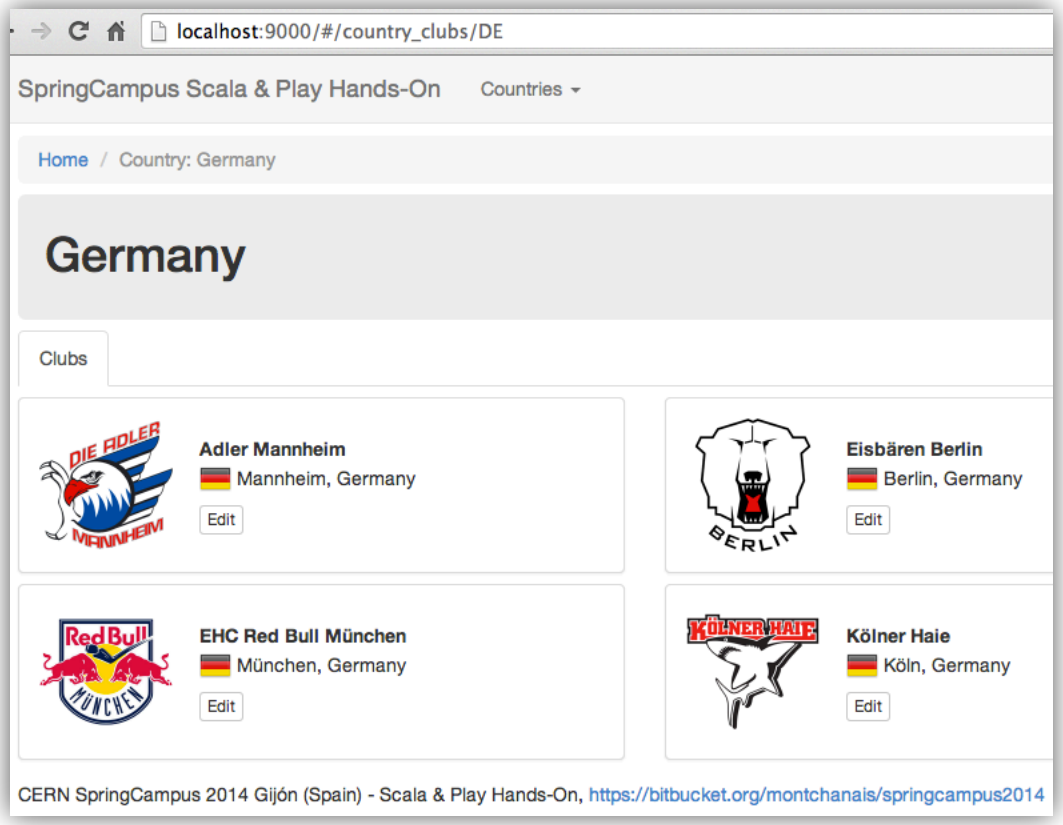
Agenda

- Build a simple web app with Play in 5 steps
  1. Project setup
  2. Web application skeleton
  3. Persistence support
  4. Display some server fetched data
  5. Add a simple form

git checkout -f step-04

Step 04 – Display Some Data

A simple example: list of hockey clubs



Notice: The club logos belong to and are under copyright of the respective hockey clubs.
They are merely used for demonstration purposes in this presentation and the accompanying sample code.

AngularJS

```
<html ng-app="springcampusApp">
  <head>
    <title>CERN SpringCampus</title>
    <meta charset="utf-8">
```

1. Link application w. AngularJS module

```
    <!-- Include all JS files at the end to accelerate page loading. -->
    <script src='/assets/javascripts/angular.min.js'></script>
    <script src='/assets/javascripts/angular-route.min.js'></script>
    <script src='/assets/javascripts/angular-resource.min.js'></script>
    <script src='/assets/javascripts/jquery-1.11.0.min.js' type="text/javascript"></script>
    <script src='/assets/javascripts/bootstrap.min.js' type="text/javascript"></script>

    <script src='/assets/javascripts/springcampus-controllers.js' type="text/javascript"></script>
    <script src='/assets/javascripts/springcampus-services.js' type="text/javascript"></script>
    <script src='/assets/javascripts/springcampus-directives.js' type="text/javascript"></script>
    <script src='/assets/javascripts/springcampus-app.js' type="text/javascript"></script>
  </body>
</html>
```

2. Load all required JavaScript files (incl. those written by us)

3. Define an 'ng-view' element
(used as placeholder for changing views)

```
<div class="container">
  <div ng-view>
  </div>
```

Application & Routes

```
var springcampusApp = angular.module('springcampusApp', [
  'springcampusControllers',
  'springcampusServices',
  'springcampusDirectives',
  'ngRoute'
]);
```

Define the application and its dependencies

```
springcampusApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/home', {
        templateUrl: '/assets/partials/home.html',
        controller: 'HomeCtrl'
      }).
      when('/country_clubs/:countryId', {
        templateUrl: '/assets/partials/country_clubs.html',
        controller: 'ClubListCtrl'
      }).
      otherwise({
        redirectTo: '/home'
      });
  }]
);
```

Define routes, link them with templates and controllers.

Controllers & Services

- Controllers
  - Define what data is in scope
  - Link between view (template) and logic
  - Must not contain DOM manipulation code

- Services
  - General application utilities
    - E.g. communication with a server, caching etc.

Directives

- Basic reusable building blocks
- Linked to a template
- Separate scope (if desired)
  - Increases encapsulation and re-usabibility
- Integrated via HTML tags
- Can contain DOM manipulation code

Directives

country_clubs.html

```
<springcampus-country-header country="country" active-tab="clubs"></springcampus-country-header>
<div class="springcampus-pagecontent">
  <div class="row">
    <div class="col-md-6">
      <div ng-repeat="club in clubsHalf1 | orderBy:'name'" style="...">
        <springcampus-club-info club="club"></springcampus-club-info>
      </div>
    </div>
    <div class="col-md-6">
      <div ng-repeat="club in clubsHalf2 | orderBy:'name'" style="...">
        <springcampus-club-info club="club"></springcampus-club-info>
      </div>
    </div>
  </div>
</div>
```

Directives build components that are included via tags.
Variables in scope or text are passed as argument values.

Also Consider …

- LESS
  - A simple yet powerful replacement for CSS
    - Reuse code by defining variables
    - Compiled to CSS automatically by Play
- Client side caching
  - For applications loading
    - The same small pieces of data over and over again
    - Given that they are not likely during user session

Agenda

- Build a simple web app with Play in 5 steps
  1. Project setup
  2. Web application skeleton
  3. Persistence support
  4. Display some server fetched data
  5. Add a simple form

git checkout -f step-05

Step 05 – Simple Form

- Involved Play constructs
  - Dedicated controller
  - Case class decribing form data
    - Good practice: Do not use domain instances directly!
  - Validation logic and formatting helpers

Play Form Template

```
@(countries: List[springcampus.core.domain.Country],
  clubForm: Form[controllers.ClubFormData])

@import controllers.ClubFormHelpers._

@main("CERN SpringCampus", countries) {
<div class="container">

  @if(clubForm.hasGlobalErrors) {
    <div class="alert alert-danger">
      <ul>
        @for( error <- clubForm.globalErrors ) {
          <li>@error.message</li>
        }
      </ul>
    </div>
  }

  <!-- Inputs and submit button go here. -->

</div>
}
```

Template arguments

Import custom formatter for inputs

Display global errors (if there are any)

Input Fields & Submit

Generate form tag with some extra attributes

```
@helper.form(action = routes.EditClubController.clubEdited,
             'class -> "form-horizontal", 'role -> "form" ) {

  <!--<input type='hidden' value='@clubForm.data.get("id")' />-->
  @helper.inputText(clubForm("id"), 'class -> "form-control" )
  @helper.inputText(clubForm("name"), 'class -> "form-control" )
  @helper.select(clubForm("country"), Seq("CH"->"Switzerland",
    "DE"->"Germany","FR"->"France"), 'class -> "form-control" )
  @helper.inputText(clubForm("place"), 'class -> "form-control" )
  @helper.inputText(clubForm("arena"), 'class -> "form-control" )
  @helper.inputText(clubForm("longitude"), 'class -> "form-control" )
  @helper.inputText(clubForm("latitude"), 'class -> "form-control" )
```

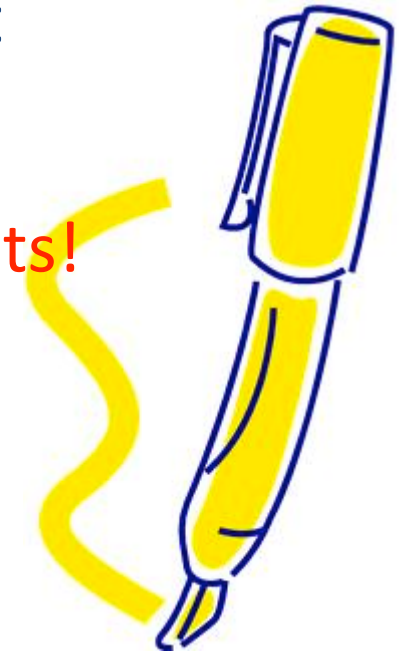
Generate input fields, specify to add special class attribute value.

```
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <button type="submit" class="btn btn-default">Save</button>
    </div>
  </div>
}
```

Add a submit button.

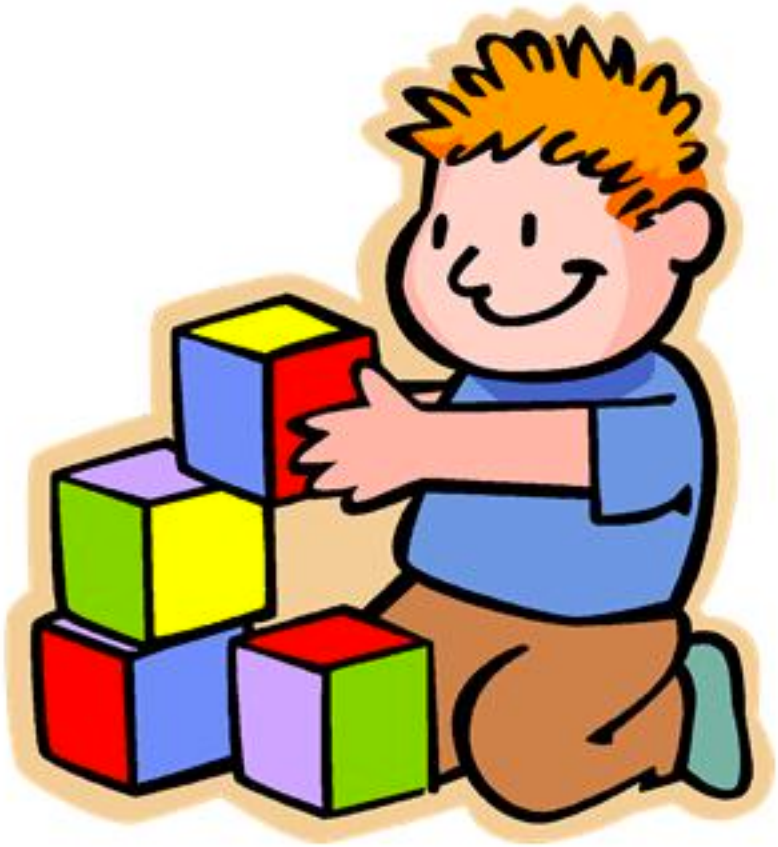
Further Play Highlights

- Content negotiation (e.g. use in REST WS)
- Filter + JSON APIs
  - Look at JSONConverter in Demo project
- Easy CSRF protection
  - Always check for CSRF attacks in requests!
- Asynchronous request handling
  - Uses Futures and their composition API
- Reverse routing



Was it a Convincing Play?

- Questions?
- Thank you!

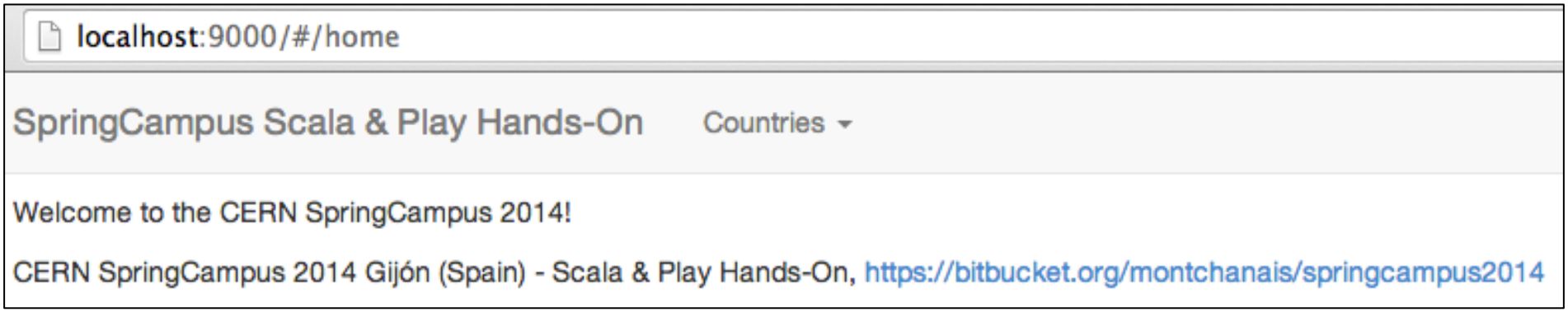


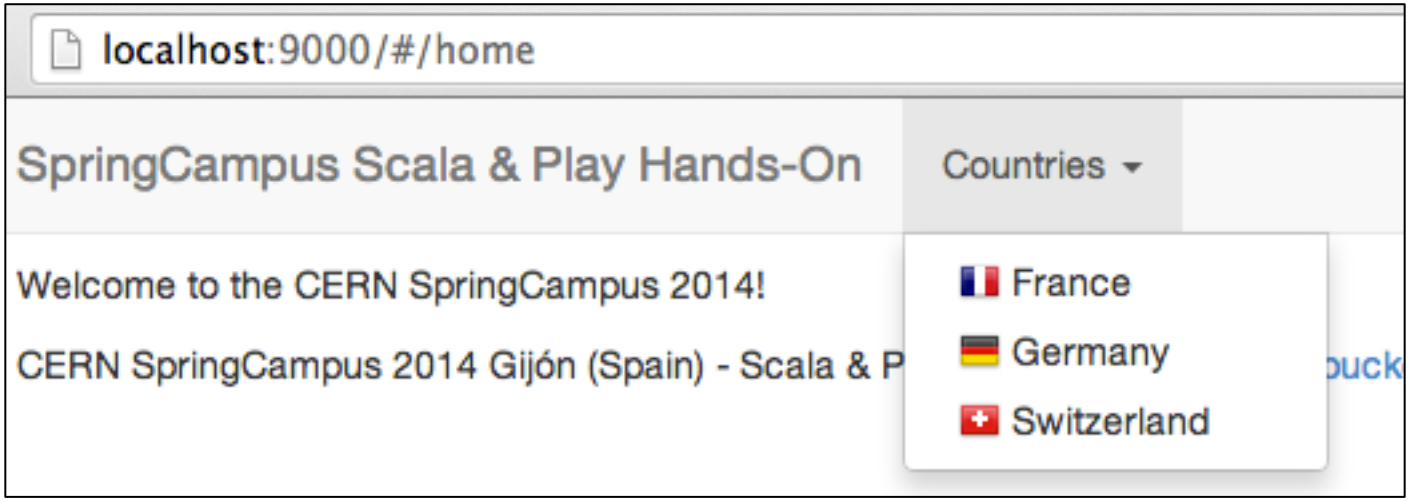
Get the sources (once the presentation is finished):
git clone https://montchanais@bitbucket.org/montchanais/springcampus2014.git

Web References

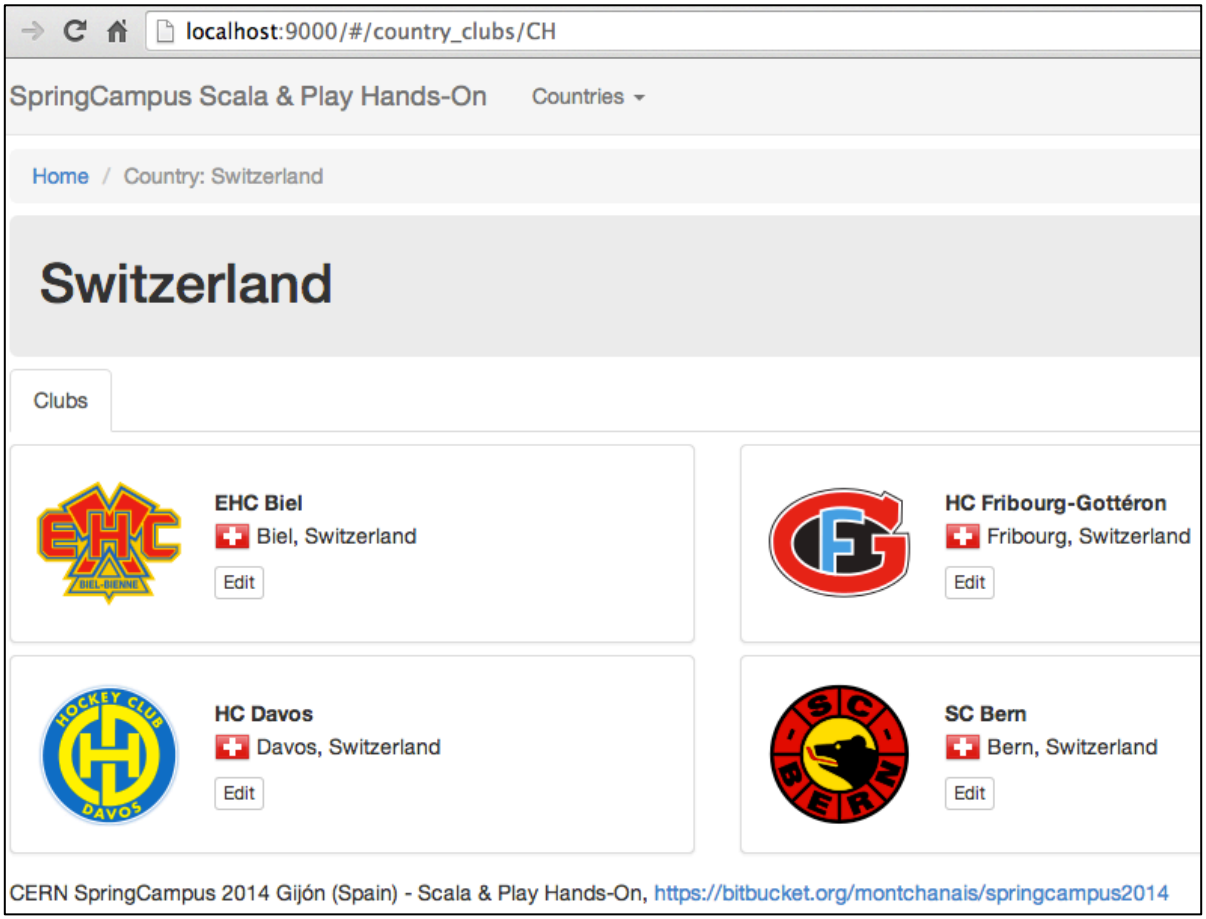
- http://typesafe.com/
- http://www.scala-lang.org/
- http://docs.scala-lang.org/
- http://www.playframework.com/
- http://angularjs.org/
- http://getbootstrap.com/
- http://jquery.com/

Screenshots: Start Page





Screenshots: List Clubs



Notice: The club logos belong to and are under copyright of the respective hockey clubs.
They are merely used for demonstration purposes in this presentation and the accompanying sample code.

Screenshots: Edit Club