Wiki

Inhoudsopgave

[IDE setup 2](#_Toc518638783)

[Intellij IDEA 2](#_Toc518638784)

[Installeren Scala plugin 2](#_Toc518638785)

[Importeren van het project. 2](#_Toc518638786)

[Play framework activeren 2](#_Toc518638787)

[Play Framework 2](#_Toc518638788)

[Aanroepen functies Java vanuit Javascript/AJAX 2](#_Toc518638789)

[Het maken van een nieuwe pagina en er naartoe navigeren 3](#_Toc518638790)

[Project setup 5](#_Toc518638791)

[Security 6](#_Toc518638792)

# IDE setup

## Intellij IDEA

Met intellij kun je heel gemakkelijk Scala projecten openen. Hieronder zijn een paar verplichte stappen om te zorgen dat Intellij het project goed oppakt. Voordat je begint met het project te importeren, volg eerst de volgende stappen.

## Installeren Scala plugin

Ik verwijs hiervoor graag naar de officiële documentatie zodat deze documentatie niet outdated raakt:

https://www.jetbrains.com/help/idea/installing-updating-and-uninstalling-repository-plugins.html

Zoek hier naar de 'scala' plugin en installeer deze.

## Importeren van het project.

Clone deze github repository en importeer het project via de build.sbt.

Intellij vraagt vervolgens om verschillende data, vul in hoe je het zelf wilt (Ik raad aan om .idea project based te blijven gebruiken.

## Play framework activeren

Intellij deactiveert automatisch play op een project, deze moet je dus zelf activeren.

Ga in je project naar File > Settings.

Ga vervolgens naar Languages & Frameworks > PLay2

Activeer vervolgens de checkbox 'Use Play 2 compiler for this project'. Druk op Ok.

Nu kan je de website runnen. Druk met je rechter muis knop op build.sbt en selecteer: 'Run Play 2 App'.

# Play Framework

## Aanroepen functies Java vanuit Javascript/AJAX

Dit zal ik als voorbeeld laten zien. Stel we willen een nieuwe functie maken die een nieuwe user aanmaakt.

**Stap 1**

Voeg de nieuwe functie toe aan het conf/routes bestand.

```

POST /gebruikers/new/:naam controllers.AccountController.newUser(naam: String)

```

Ik ga er even vanuit dat de functie ‘newUser()’ al gemaakt en implemented is.

**Stap 2**

Voeg de functie die je net hebt aangemaakt aan de jsRoutes.

Ga hiervoor naar \_/app/controllers/Application.java\_

Hier vind je een functie ‘JavascriptRoutes()’. Hieraan moet je de nieuwe functie toevoegen. 'routes.javascript.' is altijd het zelfde, Wat erachter na komt is de functie die je moet aanroepen.

```Java

public static Result javascriptRoutes() {

response().setContentType("text/javascript");

return ok(

Routes.javascriptRouter("jsRoutes",

routes.javascript.OpdrachtenController.checkAntwoord(),

routes.javascript.AccountController.newUser()

)

);

}

```

\*\*Laat hierbij altijd de parameters leeg.\*\*

**Stap 3:**

Roep de functie aan met ajax:

```Javascript

<script type="text/javascript">

jsRoutes.controllers.AccountController.newUser("nieuw\_accout").ajax({

dataType: "json",

success : function(data) {

console.log(data)

}

});

</script>

```

De datatype kan je veranderen.

## Het maken van een nieuwe pagina en er naartoe navigeren

Als voorbeeld maken we een nieuwe pagina genaamd voorbeeld.scala.html

**Stap 1**

Eerst maak je een nieuw bestand in de map view.

```

@()

<div>

Hello world

</div>

```

**Stap 2**

Stap 2 is een controller maken voor deze pagina in de controllers map

```

public class VoorbeeldController extends Controller{

public static Result voorbeeld(){

return ok(newmain.render("Statistieken", voorbeeld.render()));

}

}

```

**Stap 3**

Nu gaan we in conf/routes een regel toevoegen

```

GET /voorbeeld controllers.VoorbeeldController.voorbeeld()

```

**Stap 4**

Voor het linken naar een pagina zal er een knop moeten worden gemaakt

```

<a href="@routes.VoorbeeldController.voorbeeld()"><button class="btn btn-primary">Voorbeeld pagina</button></a>

```

**Extra**

Wanneer je op de pagina data gebruikt uit de database zal je die data moeten meesturen.

In stap 1 heb je een pagina met @().

Binnen de haakjes specificeer je wat deze pagina nodig heeft om te laden.

voorbeeld:

```

@(gebruikers : List[Gebruikers])

```

voorbeeld 2:

```

@(datums : List[AnalyzeSpellingRegel], gebruiker : Gebruiker)

```

Voor de : staat de naam van wat je meestuurt, achter de : staat het type dat je meestuurt.

In de controller wijzig je het volgende:

```

public class VoorbeeldController extends Controller{

public static Result voorbeeld(){

//definieer datums en gebruiker

return ok(newmain.render("Statistieken", voorbeeld.render(datums, gebruiker)));

}

}

```

# Project setup

Ik wil in deze wiki kort uitleggen hoe de applicatie opbouw in elkaar zit en welke aparte files belangrijk zijn.

* .idea

Een folder voor Intellij om al je configuratie in op te slaan.

* app
  + **analyze**: De folder voor alle logica voor analyze
  + **Authentication**: Alle logica voor de authenticatie. Dit wordt geregeld met behulp van annotaties
  + **Controllers**: Deze worden direct geroepen via de view. Als je een actie moet uitvoeren moet je hierin een nieuwe class maken.
  + **Dao**: Alle dao’s. Oftewel alle logica voor de database. Hier worden geen checks uitgevoerd, maar alleen database operaties.
  + **Formpackage**: hierin worden verschillende form’s aangemaakt en ook de front-end checks in de back-end uitgevoerd.
  + **Model**: Alle models met behulp van Hibernate.
  + **Views**: Alle html bestanden die met behulp van het play framework worden gevuld.
* **conf**
  + **application.conf**: applicatie configuratie van play framework.
  + **routes**: Alle routes beschreven die via het front-end kunnen worden aangeroepen.
  + **META-INF/persistance.xml**: Hibernate config bestand.
* **Project**: Verschillende bestanden voor scala/play
* **Public**: Alle bestanden die publiek kunnen worden weer gegeven. Hier komen de plaatjes, css, js etc bestanden

**Build.sbt**: De build conf bestand van Scala.

Ook zit er in het project verschillende ‘target’ folders. Mocht het lijken dat de applicatie niet meer goed cached, is het goed om deze folders te verwijderen.