Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Advanced Communication and Visualization Using Play Framework with Scala (Project B)

Faculty of Electrical Engineering and Information Technology (EIT)

MSc. Sensor Systems Technology

Prof.Dr.rer.nat. Thorsten Leize

Victory J.M. Uzochukwu

Mat. Number: 66116

uzvi1011@hs-karlsruhe.de

Abstract

In this paper, advanced communication and visualization is discussed. In particular is routing using Scala Play Framework. Java is sufficient for this task, but in this paper, and the whole project, Scala is discussed and considered. During the setting up, programming, and debugging stages of the web application ("Liste Sekretariat" web page) included in this project, some observations were made and reported. As one goes down the pages they will come to see all the observations as they are carefully highlighted in the below sections. Scala being a potent programming language, played an important part in this project. How Scala was used to achieve some functions (Scala.js, for handling JavaScript click events and manipulating of the DOM elements; also for the "UserController.scala" class, for setting up of methods that allow us access to our different view templates was achieved within this class) are also discussed in the below sections as well. This project also demanded access to a database. Postgres database was considered and is also discussed in the below sections as well, including the setup. There are other details found within this report that are relevant to the whole task which can also be found as one continues to the next sections.

1.0 Introduction

It's no more news that web development is the very front of the field of Information and Technology. Without websites, the global community may not be as informed as we are today. Information is powerful, and that's how we evolve. Web development involves developing of websites for the internet and the internet users. This development is governed strongly by certain rules, depending on the programming language the author has chosen to use during the development stage.

Play Framework is a web application framework which adheres to the model, view, and controller architectural pattern. It is written in Scala and usable from other programming languages that are compiled to JVM Bytecode, e.g. Java. But in this project, Scala is considered from start to finish.

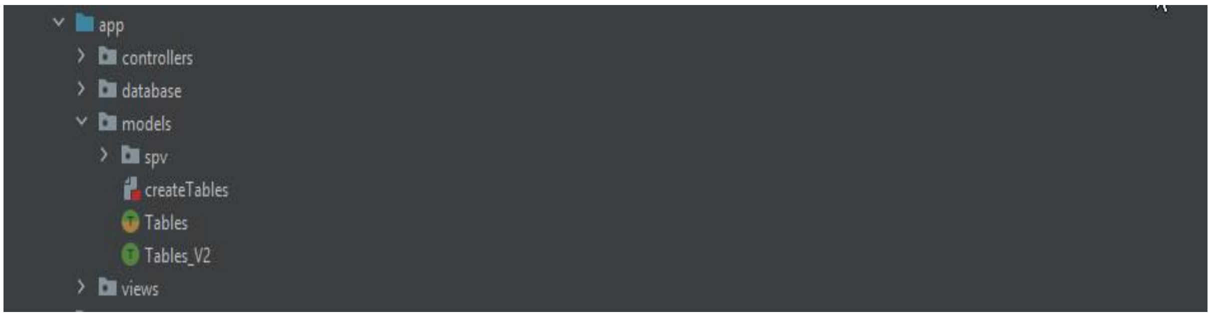
The idea here is to use Scala, the concept of Play Framework and the huge support they offer to be able to navigate across different pages, and at the same time adhering to the necessary rules required for the routing to work. Because routing allows us to select paths for traffic in our network (in our case it was done locally using localhost:9000 with AkkaHttpServer), necessary modifications were made in the routes file to be able to communicate with the controller and the views within the project. The view scripts hold the templates we have to serve to the user. To gain a stronger understanding of routing, along with Play Framework, one needs to have the necessary files and software installed on their system, and navigate to their first index.scala.html page; from there, the above will make a bit more sense.

Methods

Scala programming language was chosen as the language of choice for this project. Play framework has support for ordinary JavaScript, but Scala.js was chosen for handling click events and manipulating of the DOM elements. This is because Scala.js compiles also to ordinary JavaScript. To be able to accommodate Scala, the architectural pattern of the whole project structure had to be modified from MVC (**M**odel-**V**iew-**C**ontrol) to CSS (**C**lient-**S**erver-**S**hared). The mentioned are discussed in detail in the later sections. Gitea was chosen as the host for this project. Gitea is similar to Github, with its advanced interface, just like Github. If you have worked before with Github, you can easily adapt to Gitea. The Git command tool is also compatible with Gitea. With Git the necessary project is cloned and the necessary commits done, just like with Github. Gitea is a stable, powerful, and a reliable host for projects ranging from personal, academic, and all the way to e-commerce projects. Intellij Idea Ultimate was chosen as the development environment within which the application was written. The Java VM used was Java 11.0.8.hs-adpt during the time of this project. With the help of other referenced pages as well, this report and the project was able to be completed successfully. For this task, Postgres database was chosen. With its friendly interface, it allowed us store and manage our data efficiently.

2.0 There are some terms that are relevant to the whole project

1. **Model:**

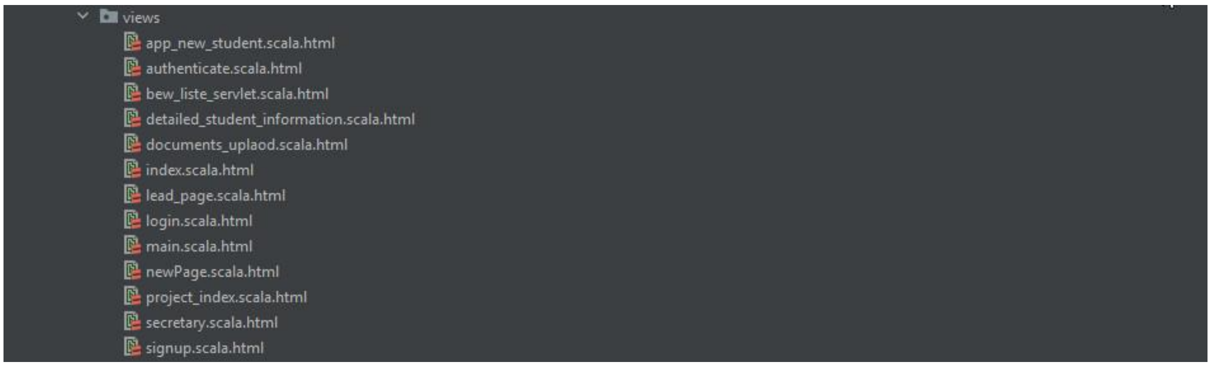


```
app
  controllers
  database
  models
    spv
    createTables
    Tables
    Tables_V2
  views
```

Fig 2.0: Model folder

A models folder contains the data necessary to the Play application. Data like the database, the tables, and the script that holds the *CodeGen* source code. When *CodeGen* is run, the necessary tables are created automatically within the specified directory.

2. **View:**

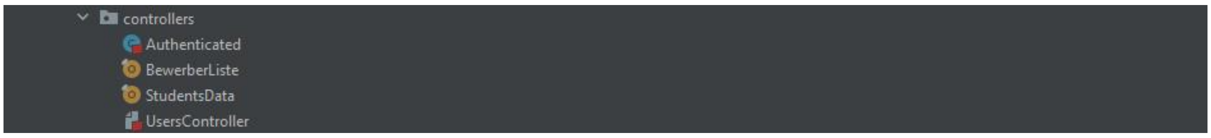


```
views
  app_new_student.scala.html
  authenticate.scala.html
  bew_liste_servlet.scala.html
  detailed_student_information.scala.html
  documents_uplaod.scala.html
  index.scala.html
  lead_page.scala.html
  login.scala.html
  main.scala.html
  newPage.scala.html
  project_index.scala.html
  secretary.scala.html
  signup.scala.html
```

Fig 2.1: View folder

The views folder contains all the information we want the user to see. In this case our template files. Every template file must end with ".scala.html", and has to be created within the views folder for a proper control from the controller class.

3. **Controller:**

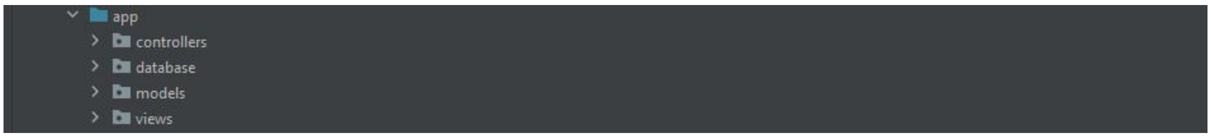


```
controllers
  Authenticated
  BewerberListe
  StudentsData
  UsersController
```

Fig 2.2: Controller folder

The controller updates the view and the model. It is within the controller that you decide which template to show to the user and how. Within the controller you may proceed to write your logic. Sessions can still be created and passed from the controller to the view. You can include a Scala singleton script and include the singleton within the controller class; this idea is good to give you more space to work with; so you don't have to write everything in the controller class. The controller is a powerful part of Play's architecture. Without the controller, nothing will work. Picture the controller the driver of a vehicle, the view the beautiful body of the vehicle, and the model the engine, break, and other details a car needs to start and move; this example may not have captured it all but it's good to have a mental picture of a concept as complex as Play Framework which aids you as you advance through the topic.

4. **App Folder:**



```
app
  controllers
  database
  models
  views
```

Fig 2.3: App parent folder

The app folder is the parent folder that houses the three major folders discussed above. By the time of this project one could create other folders within the parent folder without affecting the build process of the entire project. You could create as many folders as your application demanded within the app folder. But it's god to keep it short and neat.

5. **Simple Build Tool (.SBT):** this tool is quite a potent tool in Play; it allows the programmer perform such function as initializing their Play application. Simply type into the terminal: "sbt run" so long as all the setups have been carefully done without further bugs; with this command, you initialize your play application.

6. **Postgres Database:** for this task, Postgres database was chosen. Postgres database with its data management capabilities allowed us to store all the data relevant to this project. MySQL is sufficient for this task as well, but Postgres was considered for this purpose.

7. **Route:** allows us set up paths for traffic in our network. Within thin routes script, the programmer sets up all the relevant paths to different locations within their application. Routing is another powerful concept that enables different pages found within the application to be managed neatly and efficiently. Just like every other script, there are rules to adhere to when working on the routes script. The programmer obviously gets a warning when they have not adhered to these rules (the controller class must exist; the methods you are invoking must exist within that controller class; arguments must be passed accordingly, and so on).

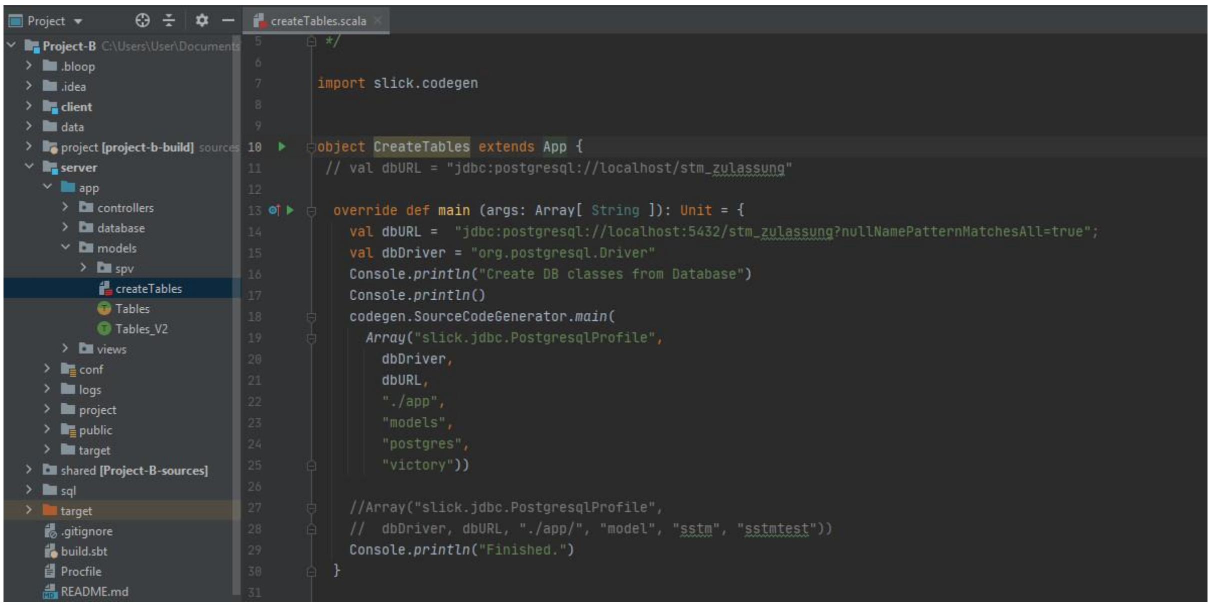
8. **CSRF:** Cross-site request forgery, also known as one-click attack or session riding and abbreviated as CSRF (sometimes pronounced *sea-surf*[1]) or XSRF, is a type of malicious exploit of a website where unauthorized commands are submitted from a user that the web application trusts. [2] For your post request to go through, you have to specify this within you view script, within that form you plan to submit. This information is very important if your post request will go through.

9. **Akka:** is a free and open-source toolkit and runtime simplifying the construction of concurrent and distributed applications on the JVM. Akka supports multiple programming models for concurrency, but it emphasizes actor-based concurrency. [3]

3.0 Play Framework

Play Framework is an open-source web application framework which follows the model–view–controller (MVC) architectural pattern. It is written in Scala and usable from other programming languages that are compiled to JVM Bytecode, e.g. Java. It aims to optimize developer productivity by using convention over configuration. [1] The "convention over configuration" part of it reduces the amount of decision the programmer has to make each time they work with the framework without losing flexibility. This idea saves more time for the programmer. Time is very important in computer programming. Any programming language or framework that consumes more time than necessary may find it a bit hard to compete with other programming languages, in terms of popularity. Python programming language grows rapidly even as I write. One can attribute such rapid growth to the speed with which the output is displayed, among other features that make Python stand out in its unique way.

The MVC model has been introduced in the previous chapter. This chapter will aim to discuss the structure of Play Framework. There are other sufficient literature out there that go even deeper into the concept of Play Framework; these materials are great if one hopes to advance even further with Play Framework. The below project structure is not the original structure of Play Framework. The original structure has just the MVC pattern. One can easily notice that the structure below is of more complex pattern with the **client, server,** and **shared** folder included. This modification was unavoidable to be able to accommodate Scala.js. This structure is further discussed in the later section.

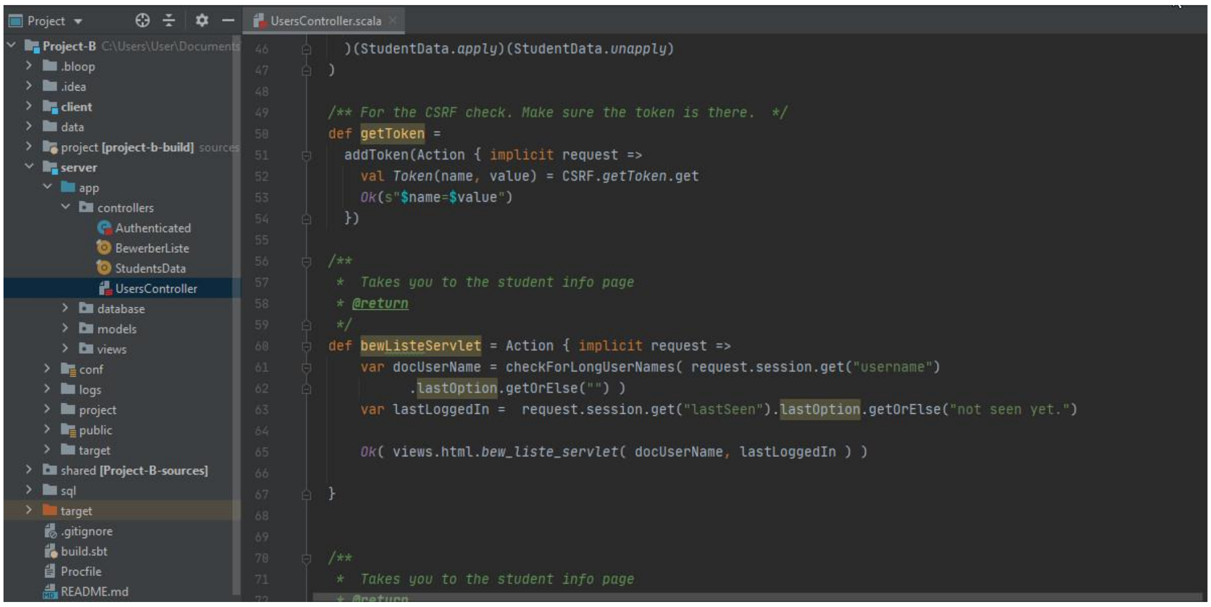


```scala
*/

import slick.codegen


object CreateTables extends App {
 // val dbURL = "jdbc:postgresql://localhost/stm_zulassung"

  override def main (args: Array[ String ]): Unit = {
    val dbURL =  "jdbc:postgresql://localhost:5432/stm_zulassung?nullNamePatternMatchesAll=true";
    val dbDriver = "org.postgresql.Driver"
    Console.println("Create DB classes from Database")
    Console.println()
    codegen.SourceCodeGenerator.main(
      Array("slick.jdbc.PostgresqlProfile",
        dbDriver,
        dbURL,
        "./app",
        "models",
        "postgres",
        "victory"))

    //Array("slick.jdbc.PostgresqlProfile",
    //  dbDriver, dbURL, "./app/", "model", "sstm", "sstmtest"))
    Console.println("Finished.")
  }
```

Fig 3.0: Model folder with create table singleton

From the above figure, the **model** folder is shown again, but this time with more information. One can easily see that "**slick.codegen"** is imported. To be able to start working with **codegen** the programmer has to include the necessary codegen dependency in the **build.sbt** script. This step is very important to be able to generate your tables. When you run this script by clicking on the green arrow head, the necessary tables are created and set ready for, maybe, user authentication or other tasks of same nature. The described is the most essential step within the model folder of the Play Framework.

The controller class must extend parent **controller** class. For this project, our controller extends **MessageAbstractController.** This extension is very import, and falls within the basic setups the programmer must make to be able to achieve a proper control of their view and model scripts.

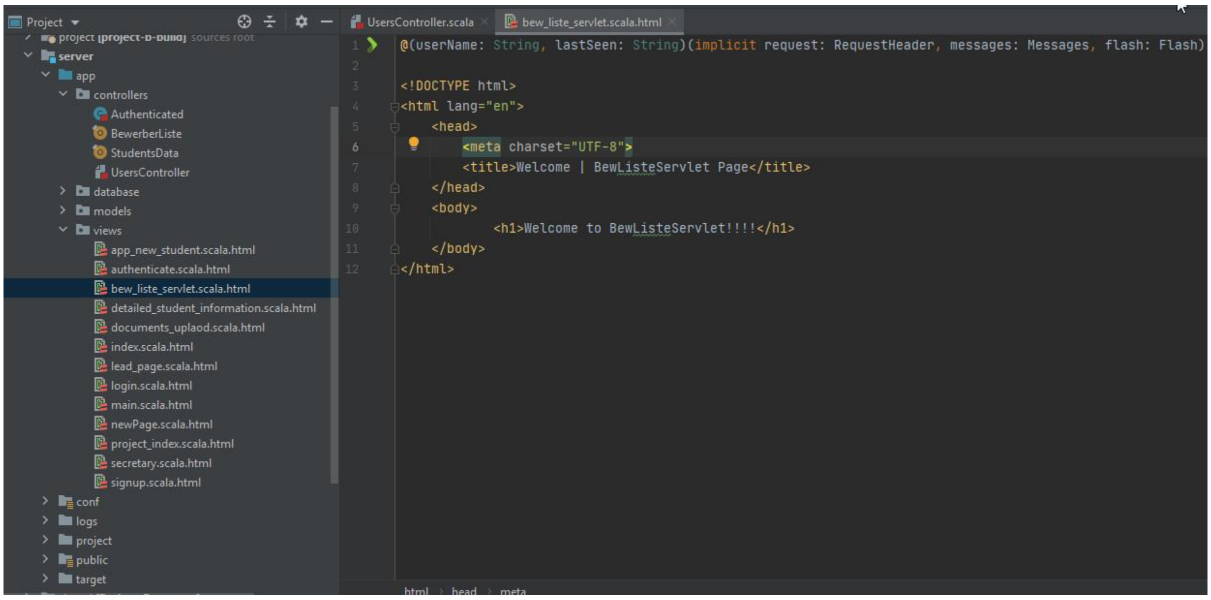


```scala
  )(StudentData.apply)(StudentData.unapply)
)

/** For the CSRF check. Make sure the token is there.  */
def getToken =
  addToken(Action { implicit request =>
    val Token(name, value) = CSRF.getToken.get
    Ok(s"$name=$value")
  })

/**
 *  Takes you to the student info page
 * @return
 */
def bewListeServlet = Action { implicit request =>
    var docUserName = checkForLongUserNames( request.session.get("username")
          .lastOption.getOrElse("") )
    var lastLoggedIn =  request.session.get("lastSeen").lastOption.getOrElse("not seen yet.")

    Ok( views.html.bew_liste_servlet( docUserName, lastLoggedIn ) )

}


/**
 *  Takes you to the student info page
```

Fig 3.1: the controller class

Notice that within the controller class, the **bewListeServlet** method returns **Ok().** Also see that the **Ok()** function takes an argument. Notice that the argument is one of our view scripts. This is the "control" in its basic form. Because this way we can "control" what the user can and cannot see. One can also see how sessions are passed to the view script to be used within our view script; in this case our **bew_liste_servlet** script. Please refer to the project to get a full grip on the contents of this **UserController** class.

The **view** script is another important part of the Play's architecture. As mention in the previous section, it contains all the information we want the user of our application to see. Basically, it is a HTML script. If you decide to copy one of your HTML codes from your ordinary text editor and paste it in this script, it will work accurately, with no detail missing. However, this particular view demands that the programmer has at least a basic understanding of Twirl templating engine. Twirl templating engine is a very powerful Scala-based templating engine inspired by ASP.NET. If the programmer has written PHP before, and worked with Smarty or Twig templating engine, this concept will not be entirely far from them.



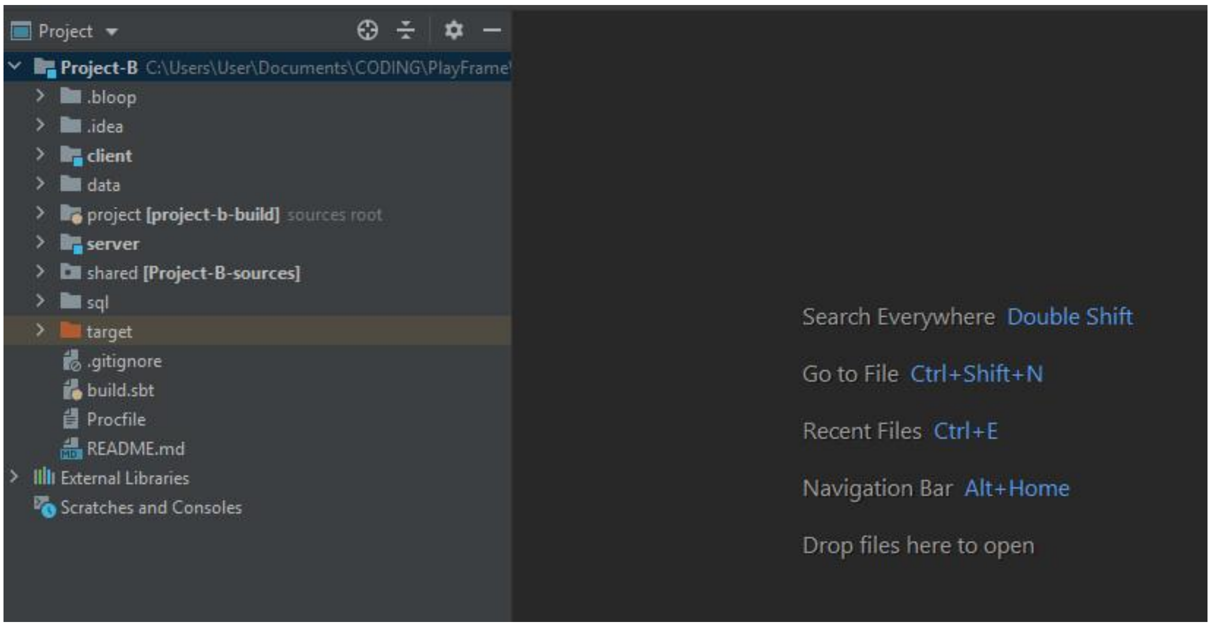
```
1   @(userName: String, lastSeen: String)(implicit request: RequestHeader, messages: Messages, flash: Flash)
2
3   <!DOCTYPE html>
4   <html lang="en">
5       <head>
6           <meta charset="UTF-8">
7           <title>Welcome | BewListeServlet Page</title>
8       </head>
9       <body>
10              <h1>Welcome to BewListeServlet!!!!</h1>
11      </body>
12  </html>
```

Fig 3.2 view template

Notice that the script contains, on the first line, some information before the HTML code. The **@(username: String, lastSeen: String)** is quite important since we specified on the controller script that we plan to pass our data from our session to this script. So to be able to get those information, which are strings, we have to inform the view script by writing as seen in the figure above. Same is done for the rest of the parameters found on line one.

4.0 Play Frame Work Using Scala.js

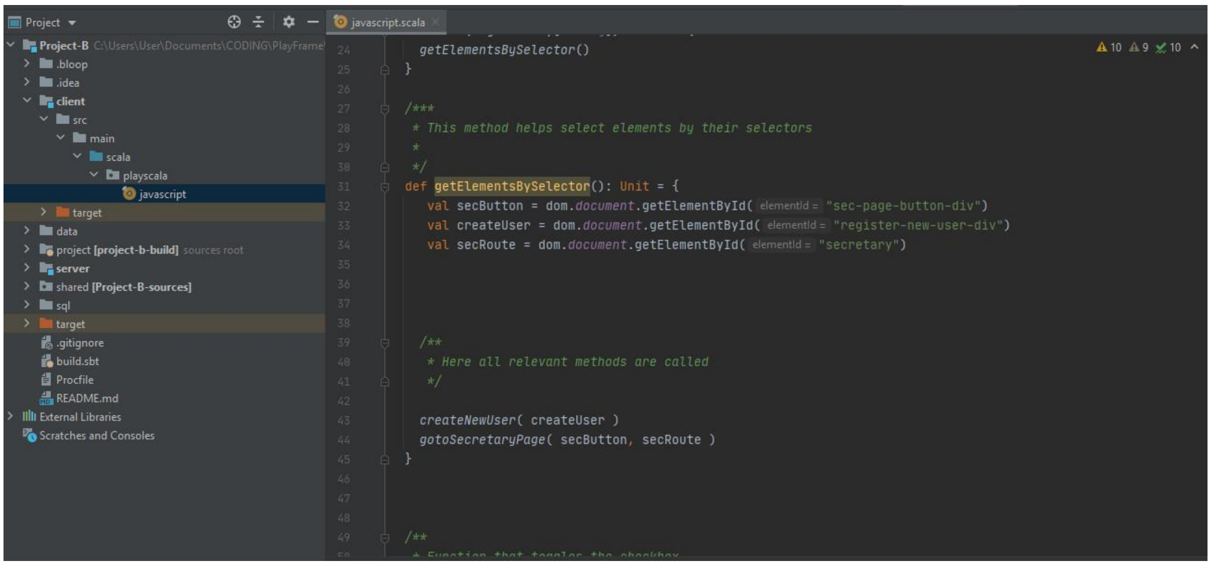
Scala is sufficient to allow a programmer write JavaScript codes within their Scala script, which are later compiled into ordinary JavaScript. But to do this, the entire structure of our Play project had to be modified to accommodate Scala.js. As mentioned before, the original pattern of Play is MVC; but for one to work with Scala.js the pattern needs to change to CSS (**C**lient-**S**erver-**S**hared) architectural pattern. This modification was unavoidable by the time this report was written. "Tomorrow and its dynamic nature." It is possible that tomorrow a new, better pattern may be introduced; but so far, this pattern got rid of all the warnings and errors that the author suffered during the initial setup as he spent a lot of effort to get the "Hello, world" displayed on the browser console. It is still possible that there is already existing pattern as by the time this report was written, but which ever one. This pattern works just fine. Other modifications were made also in the **build.sbt** folder to be able to accommodate this new structure.



```
Project
Project-B  C:\Users\User\Documents\CODING\PlayFrame
  .bloop
  .idea
  client
  data
  project [project-b-build]  sources root
  server
  shared [Project-B-sources]
  sql
  target
  .gitignore
  build.sbt
  Procfile
  README.md
External Libraries
Scratches and Consoles
```

Fig 4.0: the CSS architectural pattern to accommodate Scala.js

Your Scala.js has to be placed within the **client** folder: **client > src > main > scala > playscala >** ***yourscala.js*****.** Within this Scala script, the programmer writes the necessary programs needed to manipulate the DOM elements. One thing to notice on this script is that Scala.js preserves all the JavaScript keywords, making it so much easier for whoever that is coming from JavaScript to not be entirely thrown off by new keywords. Please see the figure below for the Scala.js reserved keywords.

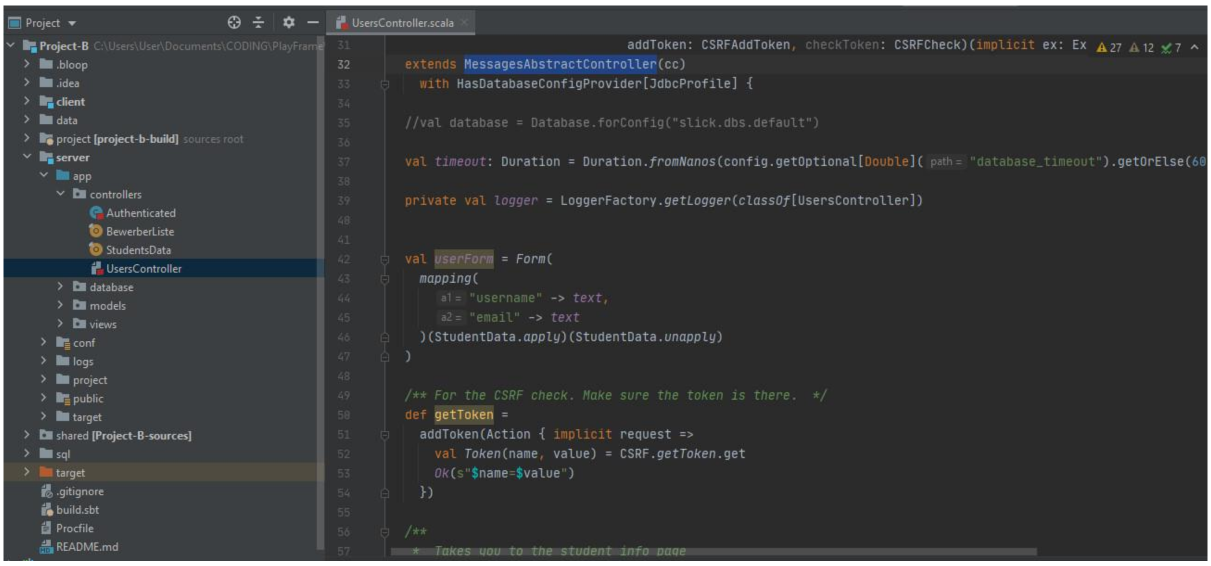


```scala
    getElementsBySelector()
}

/***
 * This method helps select elements by their selectors
 *
 */
def getElementsBySelector(): Unit = {
   val secButton = dom.document.getElementById( elementId = "sec-page-button-div")
   val createUser = dom.document.getElementById( elementId = "register-new-user-div")
   val secRoute = dom.document.getElementById( elementId = "secretary")



  /**
   * Here all relevant methods are called
   */

  createNewUser( createUser )
  gotoSecretaryPage( secButton, secRoute )
}



/**
```

Fig 4.1: our project's javascript.scala

Within this script, you can do the normal **getElementById** just like you would in ordinary JavaScript. So most of the JavaScript reserved keywords are preserved here.

The server folder is where our Play project stays. Changing this may give you error. This folder holds all the Play folders we discussed in the previous sections.

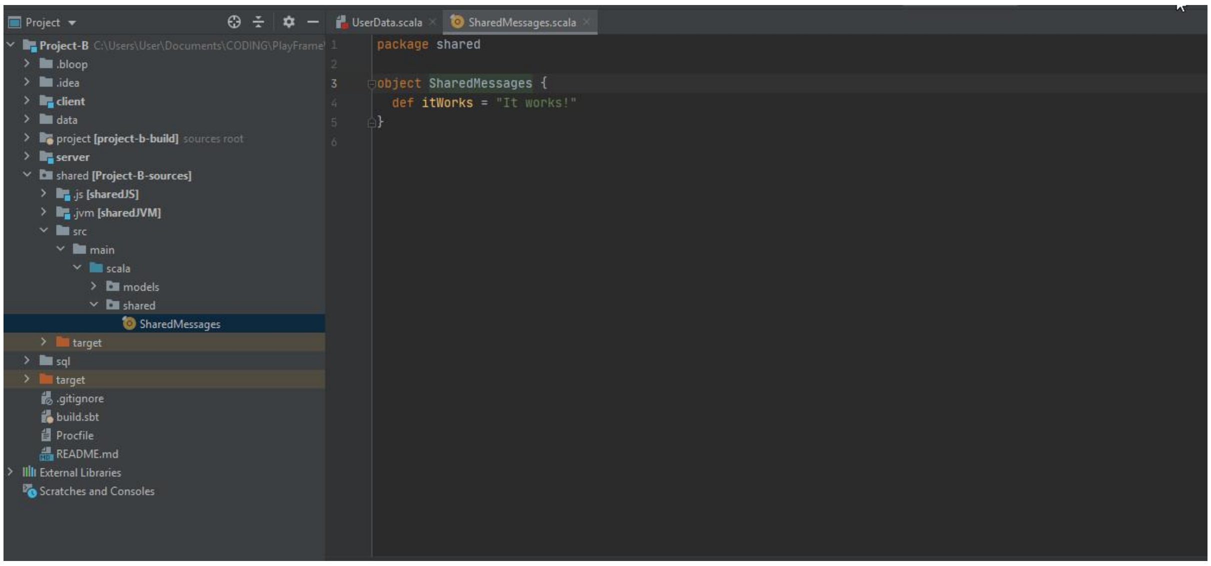


```scala
                      addToken: CSRFAddToken, checkToken: CSRFCheck)(implicit ex: Ex
extends MessagesAbstractController(cc)
  with HasDatabaseConfigProvider[JdbcProfile] {

//val database = Database.forConfig("slick.dbs.default")

val timeout: Duration = Duration.fromNanos(config.getOptional[Double]( path = "database_timeout").getOrElse(60

private val logger = LoggerFactory.getLogger(classOf[UsersController])


val userForm = Form(
  mapping(
    a1 = "username" -> text,
    a2 = "email" -> text
  )(StudentData.apply)(StudentData.unapply)
)

/** For the CSRF check. Make sure the token is there.  */
def getToken =
  addToken(Action { implicit request =>
    val Token(name, value) = CSRF.getToken.get
    Ok(s"$name=$value")
  })

/**
 *  Takes you to the student info page
```

Fig 4.2: our projects server folder.

Notice that as we adhere to this pattern, the build script is not within the server folder. The build script has to be placed outside for proper file extraction. The **.bloop** and **.idea** folders will come in once the build is successful.

The shared folder contains the files the programmer wants to share between the client and the server. This folder may be rarely opened, depending on the kind of application you are writing. The author barely opened the share folder during the development stage of this application.

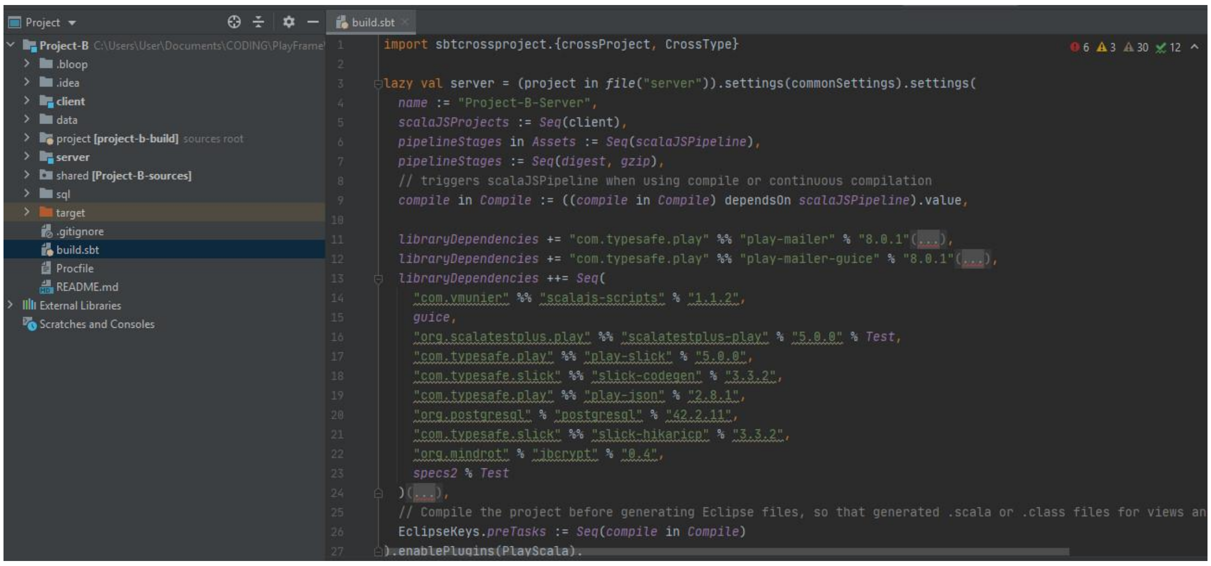


```scala
package shared

object SharedMessages {
  def itWorks = "It works!"
}
```

Fig 4.3: shared folder

4.1 Build.sbt Script

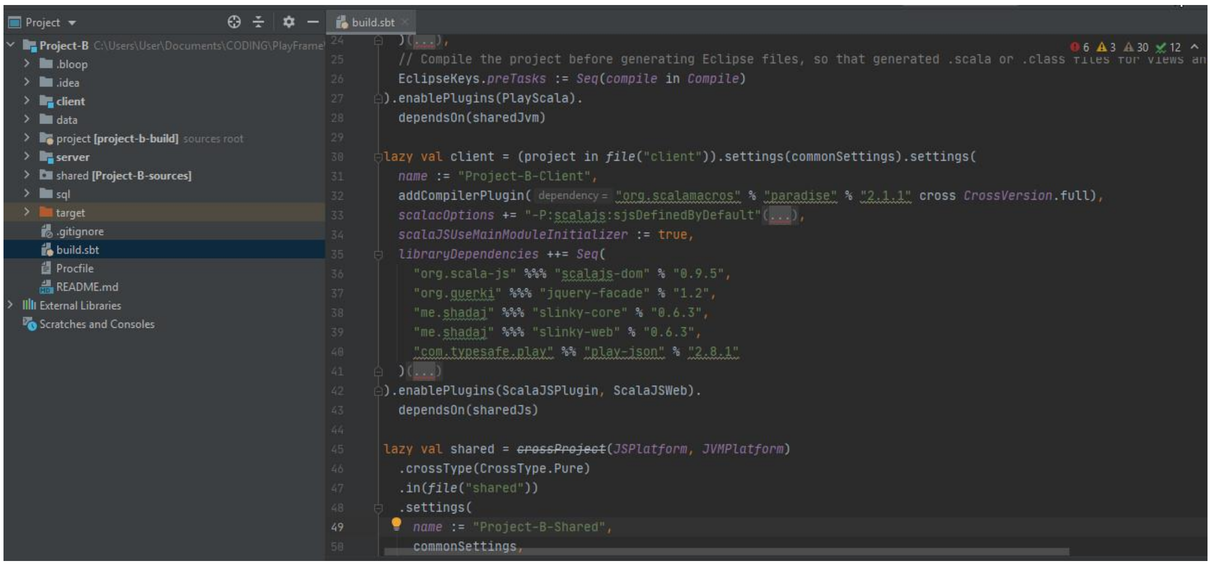
For Scala.js to be fully compatible with this project, a minor modification was made in the build script. The below figures depict these changes:



```scala
import sbtcrossproject.{crossProject, CrossType}

lazy val server = (project in file("server")).settings(commonSettings).settings(
  name := "Project-B-Server",
  scalaJSProjects := Seq(client),
  pipelineStages in Assets := Seq(scalaJSPipeline),
  pipelineStages := Seq(digest, gzip),
  // triggers scalaJSPipeline when using compile or continuous compilation
  compile in Compile := ((compile in Compile) dependsOn scalaJSPipeline).value,

  libraryDependencies += "com.typesafe.play" %% "play-mailer" % "8.0.1"(...),
  libraryDependencies += "com.typesafe.play" %% "play-mailer-guice" % "8.0.1"(...),
  libraryDependencies ++= Seq(
    "com.vmunier" %% "scalajs-scripts" % "1.1.2",
    guice,
    "org.scalatestplus.play" %% "scalatestplus-play" % "5.0.0" % Test,
    "com.typesafe.play" %% "play-slick" % "5.0.0",
    "com.typesafe.slick" %% "slick-codegen" % "3.3.2",
    "com.typesafe.play" %% "play-json" % "2.8.1",
    "org.postgresql" % "postgresql" % "42.2.11",
    "com.typesafe.slick" %% "slick-hikaricp" % "3.3.2",
    "org.mindrot" % "jbcrypt" % "0.4",
    specs2 % Test
  )(...),
  // Compile the project before generating Eclipse files, so that generated .scala or .class files for views an
  EclipseKeys.preTasks := Seq(compile in Compile)
).enablePlugins(PlayScala).
```

Fig 4.1.0: Project-B-Server

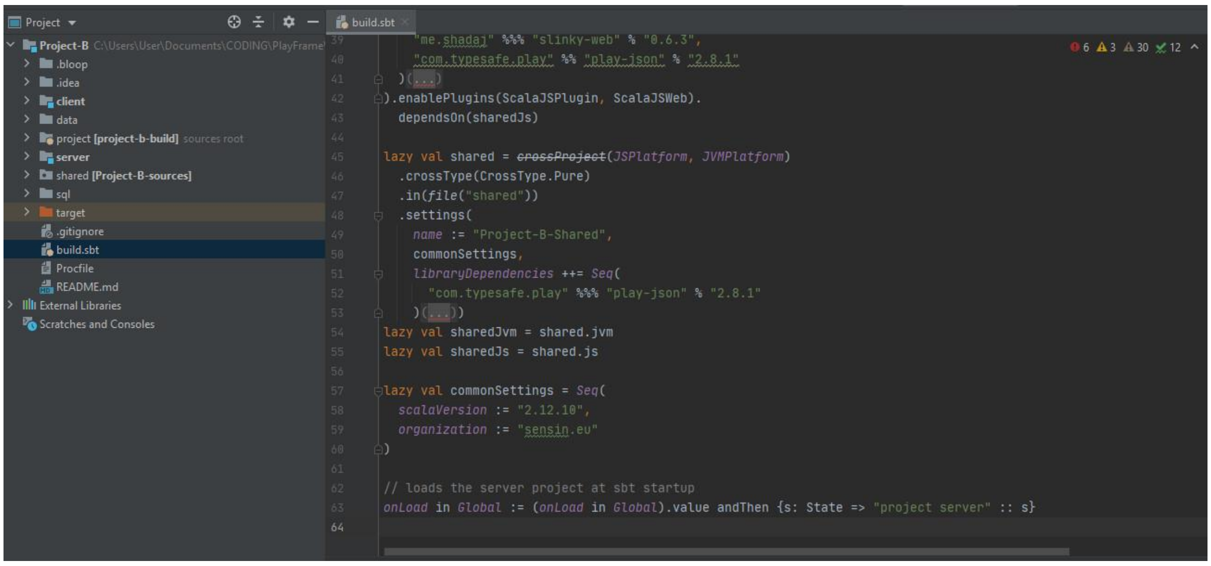
Please refer to the project itself to get a proper view of these changes. If a programmer plans to use just Play Framework with ordinary JavaScript functionalities, they don't need this setup. They just need to go to the Play's official web page and get a "Hello, world" tutorial to see the basic Play setup; but if the programmer hopes to include Scala.js, the above setup is quite important to be able to start with Scala.js. There may be other ways to set up this page, but the author's findings during his research suggested that this method is quite popular, and could be the most popular method there is.



```scala
  )(...),
  // Compile the project before generating Eclipse files, so that generated .scala or .class files for views an
  EclipseKeys.preTasks := Seq(compile in Compile)
).enablePlugins(PlayScala).
  dependsOn(sharedJvm)

lazy val client = (project in file("client")).settings(commonSettings).settings(
  name := "Project-B-Client",
  addCompilerPlugin( dependency = "org.scalamacros" % "paradise" % "2.1.1" cross CrossVersion.full),
  scalacOptions += "-P:scalajs:sjsDefinedByDefault"(...),
  scalaJSUseMainModuleInitializer := true,
  libraryDependencies ++= Seq(
    "org.scala-js" %%% "scalajs-dom" % "0.9.5",
    "org.querki" %%% "jquery-facade" % "1.2",
    "me.shadaj" %%% "slinky-core" % "0.6.3",
    "me.shadaj" %%% "slinky-web" % "0.6.3",
    "com.typesafe.play" %% "play-json" % "2.8.1"
  )(...)
).enablePlugins(ScalaJSPlugin, ScalaJSWeb).
  dependsOn(sharedJs)

lazy val shared = ~~crossProject~~(JSPlatform, JVMPlatform)
  .crossType(CrossType.Pure)
  .in(file("shared"))
  .settings(
    name := "Project-B-Shared",
    commonSettings,
```

Fig 4.1.1: Project-B-Cleint

This setup handles the client, which is the JS, part of this project. This part of this setup was found very important to be able to get the application to build and compile properly. One can easily notice all the necessary dependencies included as well. Those are the necessary dependencies needed in the project for the JavaScript to work properly.



```scala
    "me.shadaj" %%% "slinky-web" % "0.6.3",
    "com.typesafe.play" %% "play-json" % "2.8.1"
  )(...)
).enablePlugins(ScalaJSPlugin, ScalaJSWeb).
  dependsOn(sharedJs)

lazy val shared = ~~crossProject~~(JSPlatform, JVMPlatform)
  .crossType(CrossType.Pure)
  .in(file("shared"))
  .settings(
    name := "Project-B-Shared",
    commonSettings,
    libraryDependencies ++= Seq(
      "com.typesafe.play" %%% "play-json" % "2.8.1"
    )(...))
lazy val sharedJvm = shared.jvm
lazy val sharedJs = shared.js

lazy val commonSettings = Seq(
  scalaVersion := "2.12.10",
  organization := "sensin.eu"
)

// loads the server project at sbt startup
onLoad in Global := (onLoad in Global).value andThen {s: State => "project server" :: s}
```

Fig 4.1.2: Project-B-Shared

The shared configuration also has its own setup as shown above. All these setups are essential if the programmer wants to include Scala.js in the project.

Summary:

Play Framework is a powerful framework used to create web applications and other software. The framework is compatible with Scala and Java. In this project, Scala was used for the task. Routing is another important part of Play that lets the programmer define various paths for traffic within the network. We used local network (localhost:9000) for the development stage. Postgres database offers an outstanding data management features just like MySQL. Postgress played a huge part in this project, starting from the development stage, all the way to the testing stage. Scala.js compiles to ordinary JavaScript; it was Scala.js that was considered throughout the whole task. With Scala.js, the architectural pattern of Play needs to be modified to accommodate Scala.js. The MVC pattern is retained, but will be found within the **server** folder to be able to serve contents to the user properly. The build script has to also be modified to accommodate Scala.js properly.

Acknowledgement

First I want to thank God for making the whole project successful. Everything went smoothly by His Grace. I also which to thank our professor, Prof. Dr. rer. Nat. Thorsten Leize, for exposing us to this exciting topic – Advanced Web Development with Play Framework – using Scala. The journey was full of information that a student must have in order to develop in this field. Every stage of the application – setup, development, and testing – brought out the very best in us, and taught us unique things necessary to know, especially in the field of IT. Web development is quite important in the world today, but having a very competent teacher to lead you, along with your course mates, through it is even more important. Being one of his students is a privilege. I say thank you. I also thank my course mates whom I did the project with, although their own tasks differed from mine, but still discussing with them helped as well. I say thank you.

Reference:

[1] https://en.wikipedia.org/wiki/Play_Framework
[2] https://en.wikipedia.org/wiki/Cross-site_request_forgery
[3] https://en.wikipedia.org/wiki/Akka_(toolkit)