Open-Source Report: TCP Connection

Framework: Play / Java;Scala

General Information & Licensing

Code Repository	https://github.com/jweng6/CSE312_404
License Type	The Play Framework source code is released under the **Apache 2** license.
License Description	• Commercial use • Modification • Distribution • Patent use • Private use
License Restrictions	• Trademark use • Liability • Warranty

https://www.playframework.com/documentation/2.8.x/JavaWebSockets

How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created.
Play comes with two configurable server backends called Akka-HTTP and Netty server, which handle the low-level work of processing HTTP requests and responses from TCP/IP packets once the TCP socket is created. In this project, we will use Akka - HTTP server backend to meet the requirement.

Where is the specific code that does what you use the tech for? You *must* provide a link to the specific file in the repository for your tech with a line number or number range
We use the build.sbt file to construct the environment of the play framework in https://github.com/jweng6/CSE312_404/blob/main/build.sbt, on line 10, we can see the libraryDependencies will be input libraries from guice that are shown below, and on line 8, **val akkaHttpServer : sbt.librarymanagement.ModuleID = { /* compiled code */ }** is the function we used to create the HTTP server.

```
libraryDependencies += guice
libraryDependencies += "com.alibaba" % "fastjson" % "1.2.59"
```

build.sbt

Sources not found Decompile to Java

```
package play.sbt
object PlayImport extends scala.AnyRef with play.sbt.PlayImportCompat {
  val Production : sbt.Configuration = { /* compiled code */ }
  def component(id : _root_.scala.Predef.String) : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  def movedExternal(msg : _root_.scala.Predef.String) : sbt.ModuleID = { /* compiled code */ }
  val playCore : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val nettyServer : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val akkaHttpServer : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val logback : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val evolutions : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val jdbc : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  def anorm : sbt.ModuleID = { /* compiled code */ }
  val javaCore : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val javaForms : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val jodaForms : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val javaJdbc : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  def javaEbean : sbt.ModuleID = { /* compiled code */ }
  val javaJpa : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val filters : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val jcache : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val cacheApi : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val ehcache : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val caffeine : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  def json : sbt.ModuleID = { /* compiled code */ }
  val guice : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val ws : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val javaWs : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val openId : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val playTest : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val specs2 : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val clusterSharding : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  val javaClusterSharding : sbt.librarymanagement.ModuleID = { /* compiled code */ }
  object PlayKeys extends scala.AnyRef {
    val playDefaultPort : sbt.SettingKey[scala.Int] = { /* compiled code */ }
    val playDefaultAddress : sbt.SettingKey[_root_.scala.Predef.String] = { /* compiled code */ }
```

PlayImport

guice

```scala
            actorSystem – An ActorSystem that the server can use.
            stopHook – A function that should be called by the server wh
            be used to close resources that are provided to the server.

    final case class Context(
        config: ServerConfig,
        appProvider: ApplicationProvider,
        actorSystem: ActorSystem,
        materializer: Materializer,
        stopHook: () => Future[_]
    )

    Load a server provider from the configuration and classloader.
```

```scala
}

Knows how to create an AkkaHttpServer.
class AkkaHttpServerProvider extends ServerProvider {
  override def createServer(context: ServerProvider.Context): AkkaHttpServer = {
    new AkkaHttpServer(AkkaHttpServer.Context.fromServerProviderContext(context))
  }
}
```

AkkaHttpServerProvider.scala

After we input the libraries, we will have the AkkaHttpServerProvider class from source codes, now we see this class override function createServer from serverProvider to create an AkkaHttpSercer by creating a object AkkaHttpServer based on the Context parameter.

Open-Source Report: Parsing HTTP headers

Framework: Play / Java;Scala

General Information & Licensing

Code Repository	https://github.com/jweng6/CSE312_404
License Type	The Play Framework source code is released under the **Apache 2** license.
License Description	• Commercial use • Modification • Distribution • Patent use • Private use
License Restrictions	• Trademark use • Liability • Warranty

https://www.playframework.com/documentation/2.8.x/JavaWebSockets

How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created.

Play provides a function called **createServerFromRouter()** which allows us to create a server by the known router, and we can get the routers by the file routes in https://github.com/jweng6/CSE312_404/blob/main/conf/routes. After that, we can see the apply function has already defined default root directory, port, sslPort, address, mode, properties for the server.

```scala

  protected override def createServerFromRouter(
      serverConf: ServerConfig = ServerConfig()
  )(routes: ServerComponents with BuiltInComponents => Router): Server = {
    new AkkaHttpServerComponents with BuiltInComponents with NoHttpFiltersComponents {
      override lazy val serverConfig: ServerConfig = serverConf
      override def router: Router                  = routes(this)
    }.server
  }
}

```

AkkaHttpServerProvider.scala

```scala
object ServerConfig {
  def apply(
      classLoader: ClassLoader = this.getClass.getClassLoader,
      rootDir: File = new File( pathname = "."),
      port: Option[Int] = Some(9000),
      sslPort: Option[Int] = None,
      address: String = "0.0.0.0",
      mode: Mode = Mode.Prod,
      properties: Properties = System.getProperties
  ): ServerConfig = {
    ServerConfig(
      rootDir = rootDir,
      port = port,
      sslPort = sslPort,
      address = address,
      mode = mode,
      properties = properties,
      configuration = Configuration.load(classLoader, properties, rootDirConfig(rootDir), mode == Mode.Test)
    )
```

ServerConfig.scala

Where is the specific code that does what you use the tech for? You *must* provide a link to the specific file in the repository for your tech with a line number or number range

In https://github.com/jweng6/CSE312_404/blob/main/conf/routes, conf/routes is the configuration file used by the router. This file lists all of the routes needed by the application. Each route consists of an HTTP method and URI pattern, both associated with a call to an Action generator.

On line 7, The route is POST /login controllers.HomeController.login(request : Request) means that the request method is POST, the path or the URL we require from the frontend is /login. Finally we have to call the Action generator to get the function ready to use. Once the frontend sign_in.scala.html (https://github.com/jweng6/CSE312_404/blob/impl_database/app/views/sign_in.scala.html) send the form with action is /login and method is POST, we call the function login with parameter request:Request in HomeController (https://github.com/jweng6/CSE312_404/blob/main/app/controllers/HomeController.java) within the directory controllers.

```
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~

# An example controller showing a sample home page
GET      /                           controllers.HomeController.showLogin(request : Request)
POST     /login                           controllers.HomeController.login(request : Request)

# Map static resources from the /public folder to the /assets URL path
GET     /assets/*file               controllers.Assets.versioned(path="/public", file: Asset)

GET     /register                     controllers.HomeController.showRegister(request : Request)
POST    /register                   controllers.HomeController.register(request: Request)

GET     /create                       controllers.HomeController.showCreate(request : Request)
POST    /create                        controllers.HomeController.postCreate(request : Request)

GET     /join                       controllers.HomeController.showJoin(request : Request)
POST    /join                        controllers.HomeController.postJoin(request : Request)

GET     /main                       controllers.HomeController.showMain(request : Request)
```

routes

```
<form method="POST" action="/login">
    @helper.CSRF.formField
    <input class="sign-in-input-email" placeholder="Emails" type="email" id="email" name="email" required><br>
    <input class="sign-in-input-password" placeholder="Password"type="password" id="password" name="password" requ
    <input class="dark-blue-button" type="submit" value="LOGIN">
</form>
```

sign_in.scala.html

```
public Result login(Http.Request request) throws SQLException, ClassNotFoundException {
    final Form<User> loginForm = userForm.bindFromRequest(request);
    String request_email = loginForm.get().getEmail();
    String request_password = loginForm.get().getPassword();

    if (loginForm.hasErrors()) {
        String f = "false";
        logger.error("errors = {}", loginForm.errors());
        return badRequest(views.html.sign_in.render(loginForm,f, request, messagesApi.preferred(request)));
    }
    else {
        //返回的是一个true和false
        boolean check = user.login(request_email,request_password);

        //登入正确: 去main page, 并且添加connecting的session。
        if (check ==true) {
            return redirect( url: "/main").addingToSession(request,  key: "connecting",request_email);
        }
        //登入失败: 返回登入页面, 并且添加connect_fail的session。
        return redirect( url: "/").addingToSession(request,  key: "connect_fail",request_email);
    }
}
```

controllers.java

Open-Source Report: WebSockets

Framework: Play / Java;Scala

General Information & Licensing

Code Repository	https://github.com/jweng6/CSE312_404
License Type	The Play Framework source code is released under the **Apache 2** license.

License Description	● Commercial use ● Modification ● Distribution ● Patent use ● Private use
License Restrictions	● Trademark use ● Liability ● Warranty

https://www.playframework.com/documentation/2.8.x/JavaWebSockets

How does this technology do what it does? Please explain this in detail, starting from after the TCP socket is created.

Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.