

Installations- und Bedienungsanleitung

Dennis Grabowski, Julius Zint, Philip Matesanz, Torben Voltmer

Masterprojekt „Entwicklung und Analyse einer sicheren
Web-Anwendung“
Wintersemester 18/19

9. Dezember 2018



Inhaltsverzeichnis

1. Installation des HsH-Helpers	3
1.1. Mindestanforderungen	3
1.2. Installation	3
2. Betrieb	4
3. Konfiguration	6
3.1. Serverkonfiguration	6
3.1.1. Dateien größer als 200 MB hochladen	6
3.1.2. IP-Adresse auf eine Whitelist setzen, so dass sie nicht blockiert wird	6
3.1.3. Server auf anderer IP-Adresse, anderen Port oder anderer Domäne	
laufen lassen	7
3.1.4. Server von einem anderen Host als <code>localhost</code> ansteuern	8
3.1.5. Testing: reCAPTCHA automatisierbar testen	8
3.2. Datenbankkonfiguration	8
3.2.1. Verbindungsmodus	8
4. Bedienungsanleitung	10
4.1. Einstellungen	10
4.1.1. „Session Timeout“ anpassen	10
4.1.2. Passwort ändern	10
4.1.3. Zwei-Faktor Authentisierung aktivieren	10
4.2. Sessions	10
Appendix	12
A. Werkseinstellungen unserer Applikation: <code>conf/application.conf</code>	13
B. Benutzersicht für Zusatzfunktionalität „Benutzereinstellungen“	24
C. Benutzersicht für Zusatzfunktionalität „Zwei-Faktor Authentisierung“	25
D. Benutzersicht für Zusatzfunktionalität „Sessions verwalten“	26

1. Installation des HsH-Helpers

1.1. Mindestanforderungen

Um den HsH-Helper ausführen zu können, müssen Sie Java 8 installiert haben. Unterstützung für vorherige sowie spätere Java-Versionen ist nicht gegeben.

1.2. Installation

Zu dieser Installations- und Bedienungsanleitung haben Sie eine ZIP-Datei erhalten. Diese ZIP-Datei können Sie in einem beliebigen Verzeichnis extrahieren. Dabei entsteht folgende Ordnerstruktur:

```
bin
├── hshhelper
├── hshhelper.bat
├── conf
│   ├── application.conf
│   ├── secrets.conf
│   ├── logback.xml
│   ├── ip_whitelist.txt
│   └── password_blacklist.txt
├── lib
│   ├── hshhelper.hshhelper-1.0-SNAPSHOT.jar
│   └── // Alle Bibliotheken, von denen unsere Applikation abhängt
```

Im `bin`-Ordner befinden sich Startskripte für Windows sowie UNIX. Diese können auf der Kommandozeile ausgeführt werden, um die HsH-Helper Applikation zu starten.

2. Betrieb

Bevor Sie den HsH-Helper starten können, sollten Sie verifizieren, dass der Applikation ein sicheres Geheimnis vorliegt. Dieses ist in der `conf/secrets.conf` unter dem Schlüssel `play.http.secret.key` persistiert. Um ein möglichst sicheres Geheimnis zu generieren, könnten sie die Bibliothek OpenSSL verwenden.

```
\\ RSA
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -pkeyopt
    ↪ rsa_keygen_pubexp:65537 > rsa-4096-private-key.pem
\\ EdDSA - erst ab OpenSSL v1.1.1
openssl genpkey -algorithm Ed25519 -out ed25519-private-key.pem
\\ ECC
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-384 -pkeyopt
    ↪ ec_param_enc:named_curve > p384-private-key.pem
```

Quellcode 2.1: OpenSSL-Befehl zum Erstellen eines privaten Schlüssels

Das generierte Geheimnis müssen sie dann aus der jeweiligen Datei kopieren und in der `conf/secrets.conf`-Datei speichern unter dem Schlüssel `play.http.secret.key`. Wichtig ist, dass Sie auf gar keinen Fall das Secret "changeme" nennen, da sonst der HsH-Helper nicht starten wird. Danach ist der HsH-Helper startbereit.

Um dem HsH-Helper zu starten, müssen Sie sich auf der Kommandozeile ihres Betriebssystems in den Ordner bewegen, in dem Sie zuvor den HsH-Helper extrahiert haben. Mit folgenden Befehlen können Sie dann von dort aus den HsH-Helper starten:

```
// UNIX
$ ./bin/hshhelper
// Windows
> bin\hshhelper.bat
```

Quellcode 2.2: Kommandozeilenbefehle zum Ausführen der Applikation

Nach dem Start der Applikation werden Sie auf der Kommandozeile von einigen Logging-Ausgaben begrüßt. Der Server ist betriebsbereit, sobald Sie folgende Zeile in ihrem Kommandozeileninterface sehen:

```
[info] p.c.s.AkkaHttpServer - Listening for HTTP on /0:0:0:0:0:0:0:0:9000
```

Danach können Sie den HsH-Helper von ihrem Browser erreichen, in dem Sie <http://localhost:9000> ansurfen, sofern Sie die IP-Adresse / Hostname sowie Port nicht verändert haben (siehe [Kapitel 3](#))

Während des Hochfahrens der Applikation wird ein Ordner namens `logs/` in dem Applikationsverzeichnis erstellt. Innerhalb dieses Verzeichnisses befinden sich die Zugriffs-

und Applikationslogs.

3. Konfiguration

Der HsH-Helper ist an verschiedenen Punkten konfigurierbar. Die Werkseinstellungen können Sie notfalls auch dem Anhang A entnehmen, sofern Sie anders nicht mehr darauf zugreifen können. Die hier genannten Einstellungen sind Einstellungen, die wir unterstützen und dessen Sicherheit wir garantieren können. Sollten Sie aus irgendwelchen Gründen bestimmte Zusatzfeatures¹ nutzen wollen, sind Sie auf sich allein gestellt.

Für den Fall, dass Sie die Applikation mit einer komplett separaten Konfigurationsdatei ausführen wollen, so können Sie diese beim Start der Applikation auf der Kommandozeile hinzufügen, in dem Sie folgenden Parameter anhängen:

```
-Dconfig.file=/path/to/config/file.conf
```

3.1. Serverkonfiguration

3.1.1. Dateien größer als 200 MB hochladen

Falls Sie während des Betriebs des HsH-Helpers den Nutzern erlauben möchten, Dateien hochzuladen, dessen Größe mehr als 200 Megabyte beträgt, so müssen Sie innerhalb der Konfigurationsdatei 2 Einstellungen ändern. Bei diesen Einstellungen handelt es sich um die maximale Größe eines HTTP-Pakets, die das Play Framework verarbeiten darf. Rechnen Sie also sicherheitshalber ein paar Megabyte mehr dazu.

```
play.http.parser.maxDiskBuffer = 200MB  
parsers.anyContent.maxLength = 200MB
```

Quellcode 3.1: "Dateigröße und HTTP-PaketgrößeEinstellung innerhalb der Konfigurationsdatei"

3.1.2. IP-Adresse auf eine Whitelist setzen, so dass sie nicht blockiert wird

Sollten Sie wünschen, bestimmte IP-Adressen auf eine Whitelist zu setzen, damit diese nicht von unserer „Login-Firewall“ blockiert wird, so müssen sie diese IP-Adressen in die `conf/ip_whitelist.txt` schreiben. Dabei sollte jede IP-Adresse auf eine eigene Zeile geschrieben werden. Diese Funktionalität eignet sich besonders, um eine Universität freizuschalten, damit die Studenten, die gegebenenfalls mit der selben IP-Adresse ihre Seite ansurfen nicht vom HsH-Helper ausgeschlossen werden können.

¹Menge aller möglichen Einstellungen, siehe <https://www.playframework.com/documentation/2.6.x/ProductionConfiguration>

3.1.3. Server auf anderer IP-Adresse, anderen Port oder anderer Domäne laufen lassen

Sollten Sie den HsH-Helper auf einer externen Domäne, einer bestimmten IP Adresse oder einem bestimmten Port anbieten wollen, so müssen Sie in der Konfigurationsdatei folgende Einstellungen ändern, damit die Applikation auch funktional korrekt weiterläuft.

IP-Adresse und Port können Sie beim Start der Applikation anpassen, in dem Sie folgende Parameter auf der Kommandozeile hinzufügen:

```
-Dhttp.port=1234 -Dhttp.address=123.123.123.123
```

Folgend müssen Sie die Content Security Policy (CSP) anpassen. Beim Single Sign-On wird eine JavaScript-Funktion verwendet, um nur die Anmeldeinformationen des Netzdienst zu laden, zu welchem sich ein Nutzer gerade versucht zu verbinden. Dadurch garantieren wir, dass nicht alle hinterlegten Anmeldeinformationen aller Netzdienste eines Nutzers geladen werden müssen. In unserer CSP erlauben wir kein JavaScript ausser von einem bestimmten Pfad des Servers, siehe [Quellcode 3.2](#). Dazu müssen die Hostnamen / IP-Adressen in der Policy-Deklaration (`http://localhost:9000/assets/js/` / `http://127.0.0.1:9000/assets/js/`) mit den passenden Domännennamen / IP-Adressen ersetzt werden.

Zusätzlich müssen Sie erlauben, dass Clients von ausserhalb ihren Server ansurfen können, siehe [Unterabschnitt 3.1.4](#).

```
play.filters {  
  headers {  
    contentSecurityPolicy = "default-src 'self'; script-src https://www.  
      ↪ google.com/recaptcha/ https://www.gstatic.com/recaptcha/ http  
      ↪ ://localhost:9000/assets/js/ http://127.0.0.1:9000/assets/js/;  
      ↪ frame-src https://www.google.com/recaptcha/; img-src 'self'  
      ↪ data:"  
  }  
}
```

Quellcode 3.2: "Content Security HeaderEinstellung innerhalb der Konfigurationsdatei

Zum Abschluss müssen Sie sich neue reCAPTCHA-Schlüssel erstellen². Die ihnen mitgelieferten Schlüssel sind nur für die Domänen „hsh-helper.de“, „localhost“ und „127.0.0.1“ freigeschaltet. Nach Erhalt der Schlüssel müssen Sie diese in der `secrets.conf` einfügen.

²Siehe <https://www.google.com/recaptcha/admin#list>

3.1.4. Server von einem anderen Host als localhost ansteuern

Damit Clienten, die sich nicht auf ihrem lokalen Host befinden, den HsH-Helper ansteuern können, müssen Sie entweder deren Hostnamen / IP-Adressen in der Einstellung, siehe [Quellcode 3.3](#), explizit eintragen oder die Einstellung `allowed` vollständig auskommentieren, damit jeder Host Zugriff hat.

```
play.filters {  
  hosts {  
    allowed = ["localhost:9000", "localhost:19001", "127.0.0.1:9000"]  
  }  
}
```

Quellcode 3.3: Erlaubte HostsEinstellung innerhalb der Konfigurationsdatei

3.1.5. Testing: reCAPTCHA automatisierbar testen

Für den Fall, dass Sie den HsH-Helper automatisierbaren Tests unterziehen wollen, ihnen dabei aber das reCAPTCHA in die Quere kommt, besteht die Möglichkeit, ein reCAPTCHA extra für Testzwecke einzubauen. Dazu müssen Sie den geheimen Schlüssel sowie den Schlüssel der Seite in der `secrets.conf` auszutauschen. Hierfür entnehmen Sie bitte der Seite <https://developers.google.com/recaptcha/docs/faq#id-like-to-run-automated-tests-with-recaptcha-what-should-i-do> die Schlüssel, die Google anbietet, um automatisierbare Tests durchzuführen. **Vorsicht:** Bevor Sie das tun, sollten sie die alten Schlüssel sicher abspeichern. Ohne diese können Sie kein echtes reCAPTCHA verwenden.

3.2. Datenbankkonfiguration

Unsere Applikation bietet nur Unterstützung für die Datenbank `h2`. Bei anderen Datenbanken kann nicht garantiert werden, dass die Applikation überhaupt startet. Ebenso ist die Sicherheit der Applikation unter Verwendung einer anderen Datenbank nicht gewährleistet.

3.2.1. Verbindungsmodus

Mit Werkseinstellungen wird die Datenbank im Hauptspeicher persistiert (In-Memory Database). Dies erhöht die Sicherheit, da die Datenbank an den Speicher des Prozesses

gebunden ist, und somit kein anderer Prozess darauf zugreifen kann. Nachteil ist allerdings, dass dadurch bei einem Applikationsneustart oder Rechnerabsturz der gesamte Datenbankinhalt verloren geht.

Offiziell unterstützen wir nur die In-Memory Database. Sollten Sie dennoch den Wunsch haben, die Datenbank über die Grenzen einer Applikationslaufzeit zu persistieren, so müssen Sie die Konfigurationsdatei anpassen, siehe 3.4.

```
db {  
    default.url = "jdbc:h2:mem:hshhelper;mode=mysql;MVCC=false;LOCK_MODE  
    ↪ =1"  
}
```

Quellcode 3.4: Einstellung innerhalb der Konfigurationsdatei zum Anpassen des Verbindungsmodus der verwendeten Datenbank

Um die Datenbank in einer Datei zu persistieren, die für einen späteren Applikationsstart wieder verwendet werden kann, sollten Sie `default.url` auf `"jdbc:h2:file:./hsh-helper;mode=mysql"` (inkl. Anführungsstriche) setzen.

Andere, mögliche Verbindungsmodi können der `h2-Dokumentation`³ entnommen werden.

³Für weitere Verbindungsmodi siehe http://www.h2database.com/html/features.html#database_url

4. Bedienungsanleitung

4.1. Einstellungen

In Anhang [B](#) sieht man die Benutzersicht für die Zusatzfunktionalitäten „Session-Timeout anpassen“, „Passwort ändern“, „Zwei-Faktor Authentisierung“. Diese sind zusammengefasst unter „Einstellungen“ zu finden.

4.1.1. „Session Timeout“ anpassen

Bei Werkseinstellungen ist der Session Timeout auf das absolute Minimum, 5 Minuten, gesetzt. Für Nutzer, welche diesen Timeout als zu kurz empfinden, ist es möglich, ihn auf eine andere Zahl zu setzen, bis auf ein Maximum von einem Tag. Um dies zu bewerkstelligen, muss der Nutzer nur angeben, wie viele Minuten lang eine Session gültig sein sollte.

4.1.2. Passwort ändern

Sofern ein Nutzer wünscht, sein eigenes Passwort zu ändern, so kann er dies tun in dem er sein aktuelles sowie das neue Passwort eingibt. Wegen der Implementation der verschlüsselten Netzdienstanmeldeinformationen ist es leider nötig, das aktuelle Passwort einzugeben.

4.1.3. Zwei-Faktor Authentisierung aktivieren

Falls ein Nutzer zusätzliche Sicherheit wünscht, so kann er eine Zwei-Faktor-Authentisierung aktivieren. Beim Klick auf den Knopf zum Aktivieren, wird ein Nutzer auf eine andere Ansicht (siehe Anhang [C](#)) weitergeleitet, auf welcher ihm ein QR-Code angezeigt wird. Dieser QR-Code enthält ein Geheimnis, welches von einer Handy-Applikation wie [Authy](#) oder [Google Authenticator](#) gescannt werden kann. Nachdem der Nutzer diesen QR-Code mit der Applikation seiner Wahl gescannt hat, muss er sich einen neuen Aktivierungstoken generieren lassen und dieses angeben, um die Zwei-Faktor-Authentisierung für sein Nutzerkonto zu aktivieren.

4.2. Sessions

In Anhang [D](#) sieht man die Benutzersicht für die Zusatzfunktionalität „Sessions verwalten“. Hier ist es einem Nutzer möglich, vorherige erfolgreiche Logins zu betrachten sowie alle aktuell aktiven Sessions zu sehen und sofern gewünscht auch zu invalidieren. Beim

Invalidieren einer Session durch den Klick auf den Button neben der jeweiligen Session wird diese aus der Datenbank entfernt. Sollte jemand diese Session gerade verwenden, so wird dieser Verwender dieser Session ausgeloggt. Achtung: Es ist möglich, seine eigene, gerade aktive Session zu invalidieren. Dadurch loggt man sich effektiv selbst aus.

Appendix

A. Werkseinstellungen unserer Applikation:

conf/application.conf

```
# This is the main configuration file for the application.
# https://www.playframework.com/documentation/latest/ConfigFile
# ~~~~~
# Play uses HOCON as its configuration file format. HOCON has a number
# of advantages over other config formats, but there are two things that
# can be used when modifying settings.
#
# You can include other configuration files in this main application.
  ↳ conf file:
#include "extra-config.conf"
#
# You can declare variables and substitute for them:
#mykey = ${some.value}
#
# And if an environment variable exists when there is no other
  ↳ substitution, then
# HOCON will fall back to substituting environment variable:
#mykey = ${JAVA_HOME}

## Akka
# https://www.playframework.com/documentation/latest/ScalaAkka#
  ↳ Configuration
# https://www.playframework.com/documentation/latest/JavaAkka#
  ↳ Configuration
# ~~~~~
# Play uses Akka internally and exposes Akka Streams and actors in
  ↳ Websockets and
# other streaming HTTP responses.
akka {
  # "akka.log-config-on-start" is extraordinarily useful because it log
    ↳ the complete
  # configuration at INFO level, including defaults and overrides, so it
    ↳ s worth
  # putting at the very top.
  #
  # Put the following in your conf/logback.xml file:
  #
  # <logger name="akka.actor" level="INFO" />
```

```
#
# And then uncomment this line to debug the configuration.
#
#log-config-on-start = true
}

include "secrets.conf"
## Secret key
# http://www.playframework.com/documentation/latest/ApplicationSecret
# ~~~~~
# The secret key is used to sign Play's session cookie.
# This must be changed for production, but we don't recommend you change
  → it in this file.
# play.http.secret.key = "changeme"

## Modules
# https://www.playframework.com/documentation/latest/Modules
# ~~~~~
# Control which modules are loaded when Play starts. Note that modules
  → are
# the replacement for "GlobalSettings", which are deprecated in 2.5.x.
# Please see https://www.playframework.com/documentation/latest/
  → GlobalSettings
# for more information.
#
# You can also extend Play functionality by using one of the publically
  → available
# Play modules: https://playframework.com/documentation/latest/
  → ModuleDirectory
play.modules {
  # By default, Play will load any class called Module that is defined
  # in the root package (the "app" directory), or you can define them
  # explicitly below.
  # If there are any built-in modules that you want to enable, you can
    → list them here.
  enabled += StartModule

  # If there are any built-in modules that you want to disable, you can
    → list them here.
  #disabled += ""
}

## IDE
# https://www.playframework.com/documentation/latest/IDE
```

```
# ~~~~~
# Depending on your IDE, you can add a hyperlink for errors that will
  ↳ jump you
# directly to the code location in the IDE in dev mode. The following
  ↳ line makes
# use of the IntelliJ IDEA REST interface:
#play.editor="http://localhost:63342/api/file/?file=%s&line=%s"

## Internationalisation
# https://www.playframework.com/documentation/latest/JavaI18N
# https://www.playframework.com/documentation/latest/ScalaI18N
# ~~~~~
# Play comes with its own i18n settings, which allow the user's
  ↳ preferred language
# to map through to internal messages, or allow the language to be
  ↳ stored in a cookie.
play.i18n {
  # The application languages
  langs = [ "en" ]

  # Whether the language cookie should be secure or not
  #langCookieSecure = true

  # Whether the HTTP only attribute of the cookie should be set to true
  #langCookieHttpOnly = true
}

## Play HTTP settings
# ~~~~~
play.http {
  ## Router
  # https://www.playframework.com/documentation/latest/JavaRouting
  # https://www.playframework.com/documentation/latest/ScalaRouting
  # ~~~~~
  # Define the Router object to use for this application.
  # This router will be looked up first when the application is starting
    ↳ up,
  # so make sure this is the entry point.
  # Furthermore, it's assumed your route file is named properly.
  # So for an application router like 'my.application.Router',
  # you may need to define a router file 'conf/my.application.routes'.
  # Default to Routes in the root package (aka "apps" folder) (and conf/
    ↳ routes)
  #router = my.application.Router
```

```
## Action Creator
# https://www.playframework.com/documentation/latest/JavaActionCreator
# ~~~~~
#actionCreator = null

## ErrorHandler
# https://www.playframework.com/documentation/latest/JavaRouting
# https://www.playframework.com/documentation/latest/ScalaRouting
# ~~~~~
# If null, will attempt to load a class called ErrorHandler in the
  ↳ root package,
errorHandler = extension.ErrorHandler

# Declare that all requests will run through that filter
# AccessLoggingFilter takes every request and logs which client tried
  ↳ to access which endpoint
actionCreator = extension.logging.AccessLoggingAction
actionComposition.executeActionCreatorActionFirst = true

## Session & Flash
# https://www.playframework.com/documentation/latest/JavaSessionFlash
# https://www.playframework.com/documentation/latest/ScalaSessionFlash
# ~~~~~
session {
  # Sets the cookie to be sent only over HTTPS.
  #secure = true

  # Sets the cookie to be accessed only by the server.
  #httpOnly = true
  sameSite = null

  # Sets the max-age field of the cookie to 5 minutes.
  # NOTE: this only sets when the browser will discard the cookie.
    ↳ Play will consider any
  # cookie value with a valid signature to be a valid session forever.
    ↳ To implement a server side session timeout,
  # you need to put a timestamp in the session and check it at regular
    ↳ intervals to possibly expire it.
  #maxAge = 300

  # Sets the domain on the session cookie.
  #domain = "example.com"
}
```



```
flash {
  # Sets the cookie to be sent only over HTTPS.
  #secure = true

  # Sets the cookie to be accessed only by the server.
  #httpOnly = true
}
}

## Netty Provider
# https://www.playframework.com/documentation/latest/SettingsNetty
# ~~~~~
play.server.netty {
  # Whether the Netty wire should be logged
  #log.wire = true

  # If you run Play on Linux, you can use Netty's native socket
  ↪ transport
  # for higher performance with less garbage.
  #transport = "native"
}

## WS (HTTP Client)
# https://www.playframework.com/documentation/latest/ScalaWS#Configuring
  ↪ -WS
# ~~~~~
# The HTTP client primarily used for REST APIs. The default client can
  ↪ be
# configured directly, but you can also create different client
  ↪ instances
# with customized settings. You must enable this by adding to build.sbt:
#
# libraryDependencies += ws // or javaWs if using java
#
play.ws {
  # Sets HTTP requests not to follow 302 requests
  #followRedirects = false

  # Sets the maximum number of open HTTP connections for the client.
  #ahc.maxConnectionsTotal = 50

  ## WS SSL
  # https://www.playframework.com/documentation/latest/WsSSL
```

```
# ~~~~
ssl {
  # Configuring HTTPS with Play WS does not require programming. You
    ↪ can
  # set up both trustManager and keyManager for mutual authentication,
    ↪ and
  # turn on JSSE debugging in development with a reload.
  #debug.handshake = true
  #trustManager = {
  # stores = [
  # { type = "JKS", path = "exampletrust.jks" }
  # ]
  #}
}
}

## Cache
# https://www.playframework.com/documentation/latest/JavaCache
# https://www.playframework.com/documentation/latest/ScalaCache
# ~~~~
# Play comes with an integrated cache API that can reduce the
    ↪ operational
# overhead of repeated requests. You must enable this by adding to build
    ↪ .sbt:
#
# libraryDependencies += cache
#
play.cache {
  # If you want to bind several caches, you can bind the individually
  #bindCaches = ["db-cache", "user-cache", "session-cache"]
}

## Filter Configuration
# https://www.playframework.com/documentation/latest/Filters
# ~~~~
# There are a number of built-in filters that can be enabled and
    ↪ configured
# to give Play greater security.
#
play.filters {

  # Enabled filters are run automatically against Play.
  # CSRFFilter, AllowedHostFilters, and SecurityHeadersFilters are
    ↪ enabled by default.
```

```
#enabled += filters.ExampleFilter

# Disabled filters remove elements from the enabled list.
#disabled += filters.ExampleFilter

## CORS filter configuration
# https://www.playframework.com/documentation/latest/CorsFilter
# ~~~~~
# CORS is a protocol that allows web applications to make requests
  ↳ from the browser
# across different domains.
# NOTE: You MUST apply the CORS configuration before the CSRF filter,
  ↳ as CSRF has
# dependencies on CORS settings.
cors {
  # Filter paths by a whitelist of path prefixes
  #pathPrefixes = ["/some/path", ...]

  # The allowed origins. If null, all origins are allowed.
  #allowedOrigins = ["http://www.example.com"]

  # The allowed HTTP methods. If null, all methods are allowed
  #allowedHttpMethods = ["GET", "POST"]
}

## CSRF Filter
# https://www.playframework.com/documentation/latest/ScalaCsrf#
  ↳ Applying-a-global-CSRF-filter
# https://www.playframework.com/documentation/latest/JavaCsrf#Applying
  ↳ -a-global-CSRF-filter
# ~~~~~
# Play supports multiple methods for verifying that a request is not a
  ↳ CSRF request.
# The primary mechanism is a CSRF token. This token gets placed either
  ↳ in the query string
# or body of every form submitted, and also gets placed in the users
  ↳ session.
# Play then verifies that both tokens are present and match.
csrf {
  # Sets the cookie to be sent only over HTTPS
  #cookie.secure = true

  # Defaults to CSRFErrorHandler in the root package.
  #errorHandler = MyCSRFErrorHandler
```

```
}

## Security headers filter configuration
# https://www.playframework.com/documentation/latest/SecurityHeaders
# ~~~~~
# Defines security headers that prevent XSS attacks.
# If enabled, then all options are set to the below configuration by
  ↳ default:
headers {
  # The X-Frame-Options header. If null, the header is not set.
  #frameOptions = "DENY"

  # The X-XSS-Protection header. If null, the header is not set.
  #xssProtection = "1; mode=block"

  # The X-Content-Type-Options header. If null, the header is not set.
  #contentTypeOptions = "nosniff"

  # The X-Permitted-Cross-Domain-Policies header. If null, the header
    ↳ is not set.
  #permittedCrossDomainPolicies = "master-only"

  # The Content-Security-Policy header. If null, the header is not set
    ↳ .
  contentSecurityPolicy = "default-src 'self'; script-src https://www.
    ↳ google.com/recaptcha/ https://www.gstatic.com/recaptcha/ http
    ↳ ://localhost:9000/assets/js/ http://127.0.0.1:9000/assets/js/;
    ↳ frame-src https://www.google.com/recaptcha/; img-src 'self'
    ↳ data:"
}

## Allowed hosts filter configuration
# https://www.playframework.com/documentation/latest/
  ↳ AllowedHostsFilter
# ~~~~~
# Play provides a filter that lets you configure which hosts can
  ↳ access your application.
# This is useful to prevent cache poisoning attacks.
hosts {
  # Allow requests to localhost on ports 9000 (dev) and 19001 (default
    ↳ test) ports
  allowed = ["localhost:9000", "localhost:19001", "127.0.0.1:9000"]
}
}
```

```
## Evolutions
# https://www.playframework.com/documentation/latest/Evolutions
# ~~~~~
# Evolutions allows database scripts to be automatically run on startup
#   ↪ in dev mode
# for database migrations. You must enable this by adding to build.sbt:
#
# libraryDependencies += evolutions
#
play.evocations {
  # You can disable evolutions for a specific datasource if necessary
  db.default.enabled = true
  db.default.autoApply = true
}

## Database Connection Pool
# https://www.playframework.com/documentation/latest/SettingsJDBC
# ~~~~~
# Play doesn't require a JDBC database to run, but you can easily enable
#   ↪ one.
#
# libraryDependencies += jdbc
#
play.db {
  # The combination of these two settings results in "db.default" as the
  # default JDBC pool:
  #config = "db"
  #default = "default"

  # Play uses HikariCP as the default connection pool. You can override
  # settings by changing the prototype:
  prototype {
    # Sets a fixed JDBC connection pool size of 50
    #hikaricp.minimumIdle = 50
    #hikaricp.maximumPoolSize = 50
  }
}

## JDBC Datasource
# https://www.playframework.com/documentation/latest/JavaDatabase
# https://www.playframework.com/documentation/latest/ScalaDatabase
# ~~~~~
# Once JDBC datasource is set up, you can work with several different
```

```
# database options:
#
# Slick (Scala preferred option): https://www.playframework.com/
  ↳ documentation/latest/PlaySlick
# JPA (Java preferred option): https://playframework.com/documentation/
  ↳ latest/JavaJPA
# EBean: https://playframework.com/documentation/latest/JavaEbean
# Anorm: https://www.playframework.com/documentation/latest/ScalaAnorm
#
db {
  # You can declare as many datasources as you want.
  # By convention, the default datasource is named 'default'

  # https://www.playframework.com/documentation/latest/Developing-with-
    ↳ the-H2-Database

  # Uncomment this if you want your DB to persist
  #default.url = "jdbc:h2:file:./hshhelperdb;mode=mysql;MVCC=false;
    ↳ LOCK_MODE=1"
  default.url = "jdbc:h2:mem:hshhelper;mode=mysql;MVCC=false;LOCK_MODE
    ↳ =1"

  # You can turn on SQL logging for any datasource
  # https://www.playframework.com/documentation/latest/Highlights25#
    ↳ Logging-SQL-statements
  default.logSql=true
}

ebean.default = ["models.*", "policyenforcement.session.*"]
play.ebean.defaultDatasource = default

play.mailer {
  host = "smtp.web.de"
  port = 587
  ssl = no
  tls = yes
  tlsRequired = no
  debug = no
  timeout = null
  connectiontimeout = null
  mock = no
  props {
  }
```

```
}  
  
play.http.parser.maxDiskBuffer = 200MB  
parsers.anyContent.maxLength = 200MB  
hshhelper.sendmailsfrom = "HshHelper <hshhelper@web.de>"
```

B. Benutzersicht für Zusatzfunktionalität „Benutzereinstellungen“

Angemeldet als admin (Administrator) - [Einstellungen](#) - [Logout](#)

HsHelper

Suchen

Menü

Benutzer

Netzdienste

Gruppen

Dateien

Single Sign-on

Sessions

Mitgliedschaften

Administrators

All

Peter's Group

Session-Information

Session endet in 3 Minuten

Aktive Sessions: [1](#)

Quota: [4 % aufgebraucht](#) ([45.0 Byte von 1000.0 Byte belegt](#))

Benutzer-Einstellungen

Session-Timeout

Wenn es zu einer Zeitumstellung (Winter/Sommerzeit) kommt, kann es dazu kommen, dass die Session-Dauer temporär um eine Stunde verlängert wird.

Timeout (in min):

Minimum value: 5, Maximum value: 1,440, Required

Speichern

Passwort ändern

Aktuelles Passwort:

Required, Maximum length: 72

Neues Passwort:

Required, Maximum length: 72

Neues Passwort:

Required, Maximum length: 72

Speichern

Zwei-Faktor Authentifizierung

Aktivieren

© Philip, Julius, Torben, Dennis

[Datenschutz](#) · [Nutzungsbedingungen](#) · [Impressum](#)

C. Benutzersicht für Zusatzfunktionalität „Zwei-Faktor Authentisierung“

Angemeldet als admin (Administrator) - [Einstellungen](#) - [Logout](#)

HsHelper

Menü

Benutzer

Netzdienste

Gruppen

Dateien

Single Sign-on

Sessions

Mitgliedschaften

Administrators

All

Peter's Group

Session-Information


Session endet in 3 Minuten

Aktive Sessions: [1](#)

Quota: [4 % aufgebraucht](#) ([45.0 Byte](#) von [1000.0 Byte](#) belegt)

Zwei-Faktor Authentifizierung aktivieren

Scannen Sie den folgenden Code mit einer Authentifizierungsapp, die [Time-based One-time Password Algorithmus \(TOTP\)](#) unterstützt. Es können z.B. [Authy](#) oder [Google Authenticator](#) verwendet werden. Generieren Sie anhand des Codes ein Token, um die Zwei-Faktor Authentifizierung zu aktivieren.

Code: 

Aktivierungs-Token:
Required pattern: [0-9]+

Aktivieren

© Philip, Julius, Torben, Dennis

[Datenschutz](#) · [Nutzungsbedingungen](#) · [Impressum](#)

D. Benutzersicht für Zusatzfunktionalität „Sessions verwalten“

Angemeldet als admin (Administrator) - [Einstellungen](#) - [Logout](#)

HsHelper

Suchen

Menü

Benutzer

Netzdienste

Gruppen

Dateien

Single Sign-on

Sessions

Mitgliedschaften

Administrators

All

Peter's Group

Session-Information

Session endet in 5 Minuten

Aktive Sessions: [1](#)

Quota: [4 % aufgebraucht](#) ([45.0 Byte](#) von [1000.0 Byte](#) belegt)

Sessionübersicht

Aktive Sessions

Erstellungsdatum	Verwendete IP	
04.12.2018 - 13:23:21	0:0:0:0:0:0:1	Session zerstören

Letzte erfolgreiche Logins

Zeitpunkt	Verwendete IP	User-Agent
04.12.2018 - 01:23:21	0:0:0:0:0:0:1	Other: Firefox (63)

© Philip, Julius, Torben, Dennis

[Datenschutz](#) · [Nutzungsbedingungen](#) · [Impressum](#)