# 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 subsitution, 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 extraordinarly 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

}

## 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 disable, you can list them here.

#enabled += my.application.Module

# 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 = null

## 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

# 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'"

}

## 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 example.com, its subdomains, and localhost:9000.

#allowed = [".example.com", "localhost: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.evolutions {

# You can disable evolutions for a specific datasource if necessary

#db.default.enabled = false

}

## 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

#default.driver = org.h2.Driver

#default.url = "jdbc:h2:mem:play"

#default.username = sa

#default.password = ""

# You can turn on SQL logging for any datasource

# https://www.playframework.com/documentation/latest/Highlights25#Logging-SQL-statements

#default.logSql=true

}