# Javache Web Server documentation

This documentation will help you understand how to deploy applications on the Javache Web Server and how to configure your web server.

This documentation will cover.

* **Javache Overview**
* **How to Run an app**
* **App libraries**
* **Configurations**

The latest version of Javache and all required libraries will be included in the examples folder under the name **00JavacheLatest.zip**

For building an application, check out other documentation like the Summer MVC one where a step-by-step guide is made on how to create a project, import required libraries and start building your app.

# Javache Overview

Javache Web Server is a Java based application that listens for a connection on a given post, accepts a connection and sends the read data from the Socket’s InputStream to a set of request handlers. Those request handlers are Broccolina and Toyote.

The connection between Javache and the request handlers is done through an interface “RequestHandler” that is implemented in the startup class on each of the request handlers.

The request handlers are currently limited to only 2, but the idea is that more request handlers could be added so in order for Javache to know which Request Handlers to use, there needs to be a configuration file. In this case, the configuration file is called “request-handlers.ini” and inside this file we specify which .jar files represent a request handler and the order we put them in that file is the order they will intercept the request in Javache.

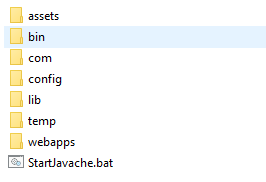
Example:

broccolina-1.1, toyote-1.1

This means the we want to load broccolina-1.1.jar from our lib folder (more on that later) and then load toyote-1.1.jar. Makes sense, because Broccolina will search for routes and if no route has matched the request route, then it will hand the request to Toyote.

The main folders in Javache:

From those folders assets and **webapps** are configurable (Their names can be changed, but more on that in the Configurations section).



Assets – place to store files. The idea is that in the **webapps** folder applications are reloaded on every start of the server so we need to store files elsewhere. Hence, in the **assets** folder. Great example for usage of the folder is when you want to store an uploaded multipart file.

Bin – The folder that contains **Javache’s dependencies**. As of version 1.0, Javache depends only on one application – **JavacheApi**, which is the application that contains the **RequestHandler** interface.



The name of the file is important for running the application (more of that in How to run the app section).

Com – This is the compiled output.

Config – This is where the configuration files are stored . As of version 1.0, there are 2 files inside.



Read more about config.ini In the **Configurations** section.

Lib – This is where you put the **request handlers** as well as the **application** libraries. Request handlers should be in that folder (Broccolina and Toyote). More about libraries in the **App Libraries** section.

Temp – This is where Broccolina stores temporary uploaded multipart files. (Files are removed on every request).

Webapps – This is where you put your applications. You can have many apps loaded at the same time. The main app is ROOT.jar by default, but this is **configurable**.

StartJavache.bat – Contains the command for starting the server on **WINDOWS**! It calls the main class and adds the Javache-api jar file in the classpath.

@echo off

java -cp ".;./bin/javache-api-1.2.6.jar;" com.cyecize.StartUp **8000**

PAUSE

In that command you can specify the port, which his **8000** by default.

# How To Run an App?

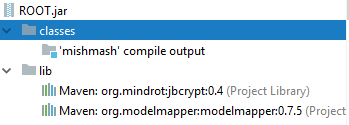
Javache allows you to run multiple applications at the same time. The folder where you want to have your apps is **webapps** (**configurable**). Inside you can put your application.jar file where the main app is by default **ROOT**.jar (**configurable**).

## How should your application jar look like?

### One way

By default, your jar file should contain a folder called “classes” and inside that folder you should put your **compiled output**. This means all classes and all resources.

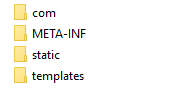
Example jar file structure.



Those two images are taken from the example with Summer MVC, which is in the Examples folder.

Inside lib folder there are .jar files that your application might depend on. Read **App Libraries** for additional information.

Inside classes folder is the compiled output, example again from the same app:



### The easiest way

The easiest way is to run mvn package. For libraries you could create a fat jar. This will NOT work by default on Javache. You need to change app\_compile\_output\_dir\_name: to “.” From “classes”.

This method is certainly easier, but the convention is that compile output should be in a folder named classes.

## Application Start Process

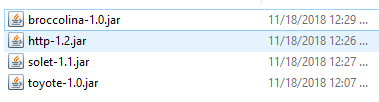
You can click on the StartJavache.bat file or execute the command inside and the application should start. If you application name is different than then main app name (ROOT by default), you will have to access you application like so: <http://localhost:8000/appName/> instead of <http://localhost:8000/>.

# App Libraries

By default, every Javache application depends on those applications: HttpAPI, SoletAPI and might depend on Summer MVC. Also it could depend on any third party library or framework like Hibernate or MySql.

There are 4 jar files that must be in Javache’s lib folder at ANY TIME!

* Broccolina
* Toyote
* HttpApi
* SummerAPI



NOTE that all names except for Broccolina and Toyote DO NOT MATTER (Javache will just add them to the classpath).

You have 4 ways to approach adding library:

* Create fat jar.
* Include the libraries in the application’s lib folder.
* Include the libraries in Javache’s lib folder.
* Both at the same time.

## Create a fat jar

Using maven and the share plugin you can create a fat jar having all or some dependencies included in the build.

A nice and easy method since you don’t need to create custom jar file. The drawback is that if you have multiple apps that use the same library (WHICH IS ALWAYS THE CASE because of HttpAPI and SoletAPI), then those libraries will be duplicated when you run Javache since Javache adds the folder if the app to its classpath.

## Include in application’s lib folder.

Similar to fat jar, but you actually put the .jar files and not the extracted classes. This is a preferred way over fat jar since you can put your compile output in a classes named folder, but it still has the same drawback – Duplication of libraries if more than 1 apps are present.

## Include in Javache’s lib folder.

You can put the application dependencies in Javache’s lib folder where those libraries will be accessible by all applications. Therefore it is the preferred method for libraries such as HttpAPI, SoletAPI or even Summer MVC and third party libraries like MySql connector. This method doesn’t really have drawbacks since you save space by adding them to Javache’s lib folder and you escape duplication, but in some cases your app might use a very rage library that is never used from other apps, which does not really make sense to be added globally.

## Both at the same time.

JUST DO BOTH. Add global libraries to Javache’s lib folder and do one of the first 2 methods for any unique dependencies for that specific app.

# Configurations

Javache has a handful of configurations, which can be found in the config/config.ini file.

Those are the possible configurations:

* max\_request\_size
* show\_request\_log
* assets\_dir\_name
* web\_apps\_dir\_name
* app\_compile\_output\_dir\_name
* main\_app\_jar\_name

### max\_request\_size

Defaults to 2 GB if not set. Specify the max size (in bytes) of each request. If the request is exceeded, you will get a white page with error 400 BAD REQUEST.

### show\_request\_log

Defaults to false. If set to TRUE, the raw request content will be printed.

### assets\_dir\_name

Defaults to assets. The name of the folder, which stores permanent data for an app.

### web\_apps\_dir\_name

Defaults to webapps. The name of the folder, which contains the applications.

### app\_compile\_output\_dir\_name

Defaults to classes. Sets the folder name for the compiled output for you applications.  
If you are using mvn package, set this to “.” Since the compile output is in the root folder.

### main\_app\_jar\_name

Defaults to ROOT.jar. The name of the main application.