# Exercises: Web Server

# Part II

Problems for exercises and homework for the [“Java Web Development Basics” course @ SoftUni](https://softuni.bg/courses/java-web-development-basics). Submit your solutions on the **course page** of the **current instance**.

This is **Part II** of the **Web Server Exercise**. The **exercise does not** come with a **skeleton**, which means you will need the **first part fully implemented** in order to do this exercise.

## Implement ResourceHandler

Create a **separate project** called **“**toyote**”** which will hold our **dynamic resource handler**. In it, **Implement** a RequestHandler.

* Create a **folder** named lib.
* Put the javache.jar in the lib folder.
* Import the javache.jar from the lib folder in the **Project**.

Create a **class** which **implements** the RequestHandler interface (you **should be able** to **implement** it if the javache.jar was **exported** and **imported** **correctly**). Name the class by your choice.

The **class** should **receive** a **Server root folder path** (as a **String**) in its **constructor**.

## Implementing the ResourceHandler Logic

Implement a **resource request handling** **logic**. Make the **ResourceHandler** search for a **resource** with the **request’s url**, in the static folder you created in the previous exercise (in the **server root**).

The **ResourceHandler** should return the **resource** as a **Response**.

The **Handler** **receives** an **InputDataStream** and an **OutputDataStream**, so you must use them in order to **handle** a **request** and **send** a **response**.

* To **extract** the **Request content** you should use the Reader from the io package.
* To **send** the **Response content** you should use the Writer from the io package.

You are free to use the HttpRequest and HttpResponse **implementations** from the http package to **ease** your work.

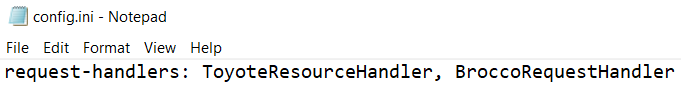
## Implementing Server Configuration

Implement a ServerConfig class on the **Server Project**, which will hold our **configuration**. The configuration will come from an **outside file**, which will be in the **server root** (**compiled files**).

The ServerConfig class should **read** the **configuration** from a file called “config.ini” in the **server root**.

For now the Server Configuration should only hold a **request handler priority configuration**. In other words, in what order should the RequestHandlers be used, in case of a client connection.

It should look like this, **for example**:



You must **load** the **data** from the config.ini file and store it in the ServerConfig class (ofcourse, upon **Server initialization**).

## Including the Server Configuration

What we just did was not meaningless. You should now use the **Server Configuration**. The Server should **pass** the **request priority configuration** onto the ConnectionHandler. The ConnectionHandler should then check one by one all RequestHandlers in the **order they are given** in the **Configuration**.

Ofcourse, if an **invalid** RequestHandlername **exists** in the **Configsuration**, it should be **skipped**.

The ConnectionHandler should pass the **Client** **data** onto the 1st **valid** RequestHandler it finds. If the RequestHandler does not intercept the connection (hasIntercepted() returns false), it should continue traversing the **RequestHandlers**.

## Polimerization

Now let’s combine everything, put the ResourceHandler in the lib folder in the **server root**.

Test the Configuration and the hasIntercepted() logic. Possible ways to do that:

* Make the RequestHandlers return **false interception** in some cases, so that the next in line can **trigger**.
* Test the Server with various **configuration orders**.
* Play around with what you’ve written. You’ve actually made quite some interesting progress.

## Implementing Solets

Go into the broccolina project and create a package solet. **Inside it** create the following **interfaces**.

Create an **interface** called HttpSoletRequest, which **extends** the HttpRequest from javache.

* Should also have a getRequestStream() – an InputStream returning method.

Create an **interface** called HttpSoletResponse, which **extends** the HttpResponse from javache.

* Should also have a getResponseStream() – an OutputStream returning method.

Create inside it an **interface** called HttpSolet. The interface should hold the following methods:

* doGet()
  + **Receives** HttpSoletRequest parameter
  + **Receives** HttpSoletResponse parameter
  + **Returns** nothing (void).
* doPost()
  + **Receives** HttpSoletRequest parameter
  + **Receives** HttpSoletResponse parameter
  + **Returns** nothing (void).
* doPut()
  + **Receives** HttpSoletRequest parameter
  + **Receives** HttpSoletResponse parameter
  + **Returns** nothing (void).
* doDelete()
  + **Receives** HttpSoletRequest parameter
  + **Receives** HttpSoletResponse parameter
  + **Returns** nothing (void).

We will need these interfaces for the Application logic we will write later on.

## Creating the WebSolet annotation

Create an **annotation** called WebSolet, which should hold a **String** **route** in itself (remember the ones we did in the Workshop?).

## Implementing Active RequestHandler Loading \*

You did a RequestHandler **loading process** in the previous exercise. But it looks kind of bad, that we need to **stop** the **Server** and **rerun** it **every time** we **change something** (even minor). Implement a **process** which **repeats every X seconds** (X > 5), and re-loads all RequestHandlers, in case of a **change**.

**HINT**: **Research** on how to do it.

## Implementing Directory Monitoring \*\*

Go an **extra step** with the RequestHandler loading process, by **implementing functionality** which **monitors** the lib **directory** for **changes** and reloads all RequestHandlers only when a **modification** / **deletion** / **creation** is **detected**.

**HINT**: It **IS** possible in **Java**.

**HINT**: **Research** on how to do it.

## Optimizing the Reloading \*\*\*

Go even further, by **implementing functionality** that **detects** **EXACTLY** which file is **modified** / **deleted** / **created** in the **monitored directory** and **reloading** only it.