BREW: Breakable web application for lectures in IT-Security

C Pohl
christoph.pohl0@hm.edu

Department for Computer Sciences and Mathematics
University of Applied Sciences
Munich

1 Introduction or how to read this file

This file looks a bit strange. There are a lot of references and broken source code. And there are some fancy references like ⟨*README* 17⟩.

At first this document (and big parts of BREW) are written in literate programming. This means you will write a documentation which includes the source code (normally you write source code and a documentation for it..). Even the makefile to generate the documentation is included in the documentation...However this is a nitty gritty way to write source code, it will also produce some uncommon documentation. To read this paper you just need to know that there are references like this ⟨*README* 17⟩. This means that there is a reference to a codechunk (a piece of code). This occurs in code chunks or in plain text. The liitle number at the end of such a reference is the page where this code is located in this document. In every code block there are one or two references. One on the left side. This is the label for this piece of code. On the right side you will find a reference to former code.

Give it a try and find the ⟨*README* 17⟩ code in this document. Than watch the README in the source directory. This is the tangling output from literate programming. Also look at the different sourcefiles, including the ⟨*Makefile* (never defined)⟩ and the LaTeXsources. Everything is build from the original *userguide.nw*.

However this is a nice way to write well documented code. In the source folder you can find the full source code, including the ⟨*Makefile* (never defined)⟩ to recreate BREW, this document and even the html version. You will need a few tools for this. At first you will need Latex, noweb (for the literate programming part), gcc, make, java and python. But this is a bit out of scope now.

2 BREW in a virtual machine

2.1 Some basics

It is not necessary to use BREW in a virtual machine. BREW is just a simple web application and a bit of source code in your IDE. However, you can use the preconfigured virtual machine.

In the most cases you will use the VirtualBox environment (You can also use KVM-qemu which is the cool way, but not really a handy solution. The KVM-qemu machine is meant for the use in a cloud environment). At first copy the .ova file. You can also test this file against the provided hashsum (You should do this if you choosed to download the full package). Howver find a suitable space in your directory and copy the .ova in this directory. Then you need to import your virtual machine. (Use import virtual machine). All the fancy configuration should be done automatically (It is preconfigured). However there are a few possible problems there.

- You need write access to the directory (Really you have to read the error statements. This is one of the most common problems in lectures...)
- You need sufficient space
- You need a 64 Bit operating system
- You need the VT flag enabled

OK what is VT flag? Read some manuals (or just google)...

OK what is 64 Bit? You are computer scientist...

Sounds trivial? OK it is! Read carefully the error messages, in most cases you will find one of the errors above....

How to deal with it? Ask your admin or use google....

2.2 Some important notes for configuration

The virtual machine is preconfigured, when you want to adopt the system to your needs read the prerequisites (c.f. Table 1) for the guest machine.

Table 1. Owasp Top Ten Attacks

Prerequisite	value
RAM	512 MB
IO-APIC	True
Processor	1
PAE-NX	False
VT-X/AMD-V	True
Graphic Ram	9MB
Network	Host only or NAT(Never Bridged...really dangerous)

Hence the most common configuration will be RAM (whatever you want) and processor (whatever you want). This will enhance the performance.

Never ever use a bridged network. BREW is a vulnerable web application. This means it is really vulnerable (no game). With a bridged network you will expose BREW to the rest of the world (or your local lan).

2.3 Start BREW and beyond

After successful import just press start in VirtualBox. The virtual machine will start and after a few seconds you will see a nice looking and well designed terminal (c.f. Figure 1)...

To log in use `username:muse` and `password:muse`. You can also login with `root:muse`. By the way, just the normal muse user has a functional X-Server configuration. In normal cases you should see something like in Figure 2

However this is nice and you are ready to start a successful career with BREW. A real nerd does not need to use stuff like the X-Server. In this moment you can use BREW just with vim and some nice console commands.

However, we had too much complaints about the usability, we implemented openbox for your convenience.

To start the X-Server type `startx` in the command line. However when everything is ok, you should see something like in Figure 3

Fig. 1. Screenshot login

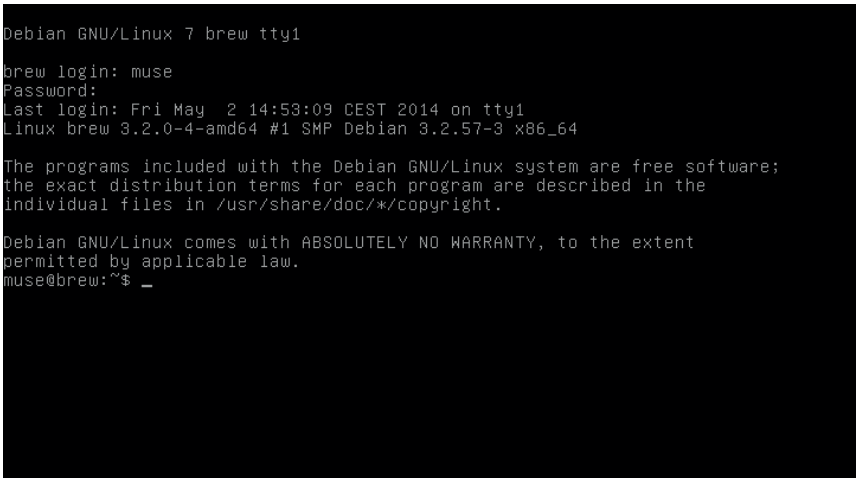

Fig. 2. Screenshot login successful

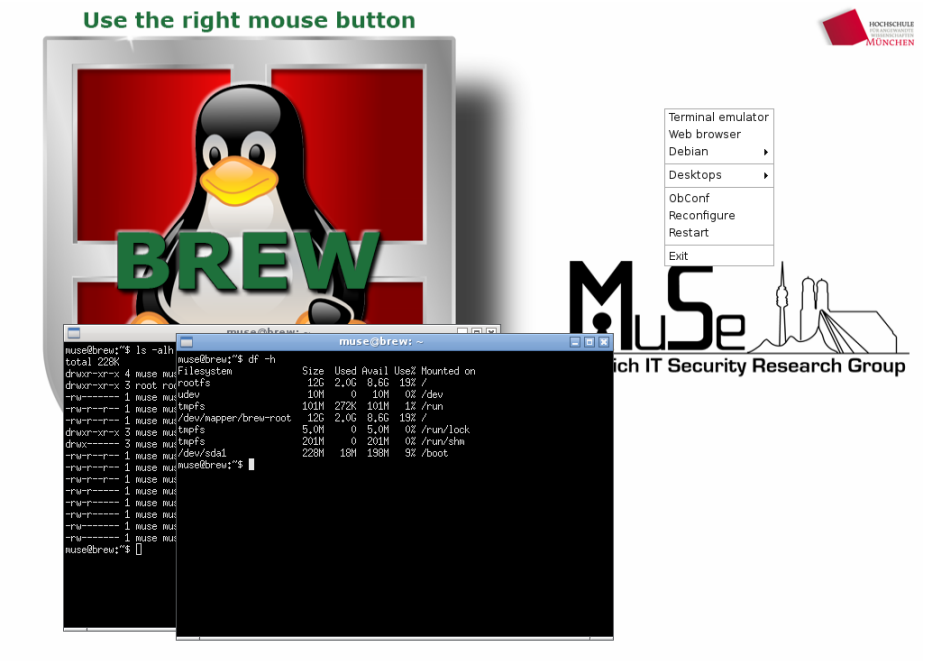

Fig. 3. BREW works

3 BREW in the IDE

BREW can be started directly from the IDE. For this you need to open the project. Do not import, extract or try other unfancy things. Just open the project. You can do this with *File→Open* from the IDE. Sounds trivial but in every lecture this is a common error... In the case of the virtual machine, you just need to open eclipse. The project is the default project.

However you will find some nitty gritty hints in the README file. Hence this guide is written in literate programming, the README file is generated out of this document... But is also part of this document (fancy or?). The file is in the appendix ⟨*README* 17⟩.

In the ⟨*README* 17⟩ you will find all necessary tips for a successful exercise.

As proposed in the ⟨*README* 17⟩ there is a mainfile. This mainfile is proposed under ⟨*TomcatServer.java* 19b⟩. When you want to change the port for BREW you can do this in this main file.

4 Architectural basics

4.1 Database and Webserver

The database is an embedded hlsql database. It always renews whenever BREW gets restarted. There is no persistent storage.

The webserver is an embedded Tomcat. It gets started whenever BREW gets started. Hence a webserver needs a listening address and port you can not start more than one server on one port...Typically you get an exception

```
...
java.net.BindException: Address already in use <null>:8081
...
```

This exception occurs whenever more than one server is bound to the same port. However this happens when you have a running instance and try to debug...

4.2 Important pathes

In BREW there are two important pathes for the exercises. The pathes are easy to find with the knowledge about the structure. As proposed in 4.3 one must find the controller and the corresponding view. Simplified at first one must look at the functionality. Each functionality (p.Ex search) has a mapping Controller (p.Ex. SearchController.java). Just look on the page in BREW, and you will find immediatly the corresponding controller. Another part of interest is the view. The view is calculated with the viewname. Simplified with `viewname+".jsp"`. In BREW you can find the controller and view on different places. The controller are under *src→edu→hm→muse→controller*, the views are within *webapps→secu→WEB-INF→pages*

4.3 Controller overview

The architecture integrates a classical MVC pattern to provide the web application.As underlying framework the widely used Spring framework is sued. The Spring framework is a lightweight platform for java applications. For the sake of simplicity, the focus is set on the vulnerabilities. A schematic overview is given in Figure 4

At first the browser send an request to BREW. Within the path matching of the Springframework a controller method gets called. In this example the search page is mapped to *http://mydomain/search.secu*. For simplicity every logical component has a related controller class (p.Ex. *Searchpage → SearchController.java*). Every action within this logical component inherits a method. (p.Ex *search → searchWebWithPost(...)*). Within this method the Businesslogic occurs. The businesslogic can ask the data access layer. This logic produces a HashMap-based model for further usage in the related view. In this example, the view gets the variable *search* accessible with the key *searchString*.

4.4 Some basics to the Controller

The Controller is just a simple javafile. It consists on different parts. Hence this is written in literate programming, we need to explain everything....

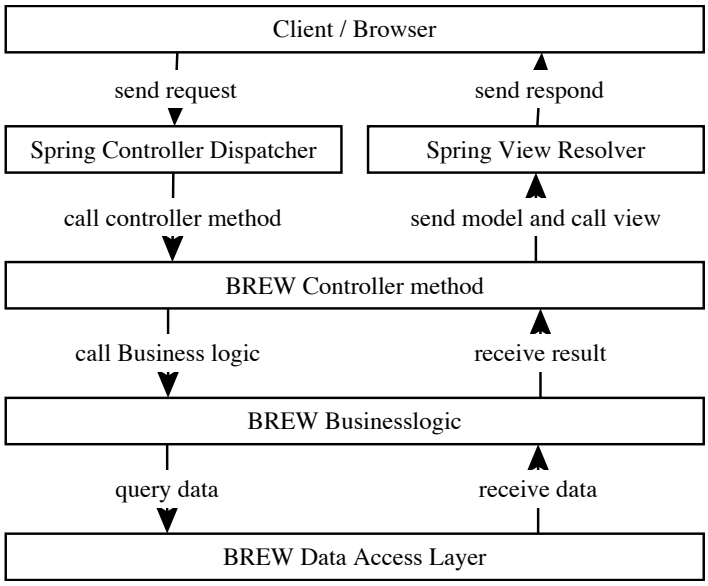

Fig. 4. BREW architecture

8a ⟨*SearchController.java* 8a⟩≡
⟨*some fancy license stuff* 18a⟩
⟨*some import statements for the SearchController* 19a⟩
⟨*class declaration for search controller* 8b⟩
⟨*method declaration for search controller* 8c⟩

Hence it is trivial to define a class. There is an important annotation. This ensures that the Spring FRamework knows that this class acts like a controller.

8b ⟨*class declaration for search controller* 8b⟩≡ (8a)
```
@Controller
public class SearchController {
```

A method for any controller is just a method. This method gets called by the dispatcher from spring. This depends on the URI and the parameter.

8c ⟨*method declaration for search controller* 8c⟩≡ (8a)
⟨*Mapping for post based search action* 8d⟩
⟨*declaration for post based search action method* 8e⟩
⟨*method body for post based seach action method* 8f⟩

The basic request mapping is done with the annotation over the corresponding method. This will map every request to the URI *http://mydomain/secu/search.secu* to this method. But only when the request method is *POST* and the parameters will match.

8d ⟨*Mapping for post based search action* 8d⟩≡ (8c)
```
@RequestMapping
        (value = "/search.secu",
        method = RequestMethod.POST)
```

The second mapping part is done over the matchin parameters. In this case there can be a parameter named *search*. The value for this parameter, p.Ex from a text input field will be present in the variable *search*. However if this parameter is not present, the variable will be null.

8e ⟨*declaration for post based search action method* 8e⟩≡ (8c)
```
public ModelAndView searchWebWithPost
        (@RequestParam
                (value = "search",
                required = false)
                String search) {
```

In normal cases the method body needs to return a Model and a View. In this body the business logic is called. The controller consists of different stages in model and view generation.

8f ⟨*method body for post based seach action method* 8f⟩≡ (8c)
⟨*generate searchmodel and view* 9a⟩
⟨*fill searchmodel with data* 9b⟩
⟨*return model and view* 9c⟩
```
}
```

To generate a model and view there is a special type in Spring. The *ModelAndView* class combines the model and view. As parameter the name of the view is injected. In our case it is *search*. This maps to *search.jsp*. Hence the mapping is generated and simplify the suffix *.jsp* gets appended.

9a ⟨*generate searchmodel and view* 9a⟩≡ (8f)
```
ModelAndView mv = new ModelAndView("search");
```

To fill the model, one must use a key and the value. With this key, the corresponding view is able to use this variable. In this case the view can ask for the value of variable *search* with the key *searchString*. In this method this is just the input from previous post request.

9b ⟨*fill searchmodel with data* 9b⟩≡ (8f)
```
mv.addObject("searchString", search);
```

Each controller needs to return a ModelAndView object. The Spring framework uses this object to render the related jsp page.

9c ⟨*return model and view* 9c⟩≡ (8f)
```
return mv;
```

4.5 Some words to the view

The views are the presentation layer in BREW. Hence there are many technologies in the wild we use simple jsp pages. This page will be called, as stated before from the controller. Simplified when the controller asks for the view *search* Spring will render the page *search.jsp*. The suffix will be appended automatically.

A view consists of different parts. Some parts are just for convienience and not for interest in our case. In our example the *search.jsp* is presented.

9d ⟨*search.jsp* 9d⟩≡
⟨*include the header and the c taglib* 9e⟩
⟨*render some nice topics* 9f⟩
⟨*present a form usable by controller* 10a⟩
⟨*render some model based output* 10b⟩
⟨*include the footer and some convienience stuff* 10c⟩

The header includes the c taglib. Look carefully for the taglib documentation. Perhaps there is something useful for the exercises..

9e ⟨*include the header and the c taglib* 9e⟩≡ (9d)
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<jsp:include page="../../head.jsp"/>
```

The topics are plain html stuff and only to present some nice look and feel

9f ⟨*render some nice topics* 9f⟩≡ (9d)
```
<h1>Search</h1>

<h2>Search</h2>
```

This is one of the more interesting parts. In the controllermethod ⟨*declaration for post based search action method* 8e⟩ the variable *search* named by the controller methodannotation *value="search"* is needed. In the form there is a form field with the name *search*. This name links to the annotated parameter in the controller. The value from this form field will be the injected parameter.

10a ⟨*present a form usable by controller* 10a⟩≡ (9d)

```
<form action="search.secu" method="post">
    <input type="text" name="search">
    <input type="submit" value="Start High Performance \
        and super safe search...">
</form>
```

However the controller also responds with a model and the inherited key value pair *searchString,value*. In the view, the pattern *${searchString}* will be replaced with this value.

10b ⟨*render some model based output* 10b⟩≡ (9d)

```
<div><c:out value="${searchString}"/></div>
 <!--
<b>You searched for ${searchString}</b>          -->
```

For a graceful end there is some convienience stuff at the end. This is just to present a nice web page without to much overhead in each jsp page.

10c ⟨*include the footer and some convienience stuff* 10c⟩≡ (9d)

```
<jsp:include page="../help/search_help.jsp"/>
<jsp:include page="../../foot.jsp"/>
```

5 The basic Makefile

However the next section is really interesting (for a nerd), it is not necessary for the lecture or to work with BREW. It is just for further reading, completeness and and some funky coding in literate programming... The following section describes how to build the makefile to make this file....

And also when you are really stucked in lecture (and want to reset BREW). However you did not listen to the lecturer, you can just use the last snapshot from the VM. This is more painless, easier and the preferred solution..

Already there? OK, following section describes the Makefile for Brew development and the userguide.

5.1 Some basics about make and the Makefile

The basic deployment for BREW is based on two Makefile. These can be used to initially install BREW (when not preconfigured), to redeploy BREW or to rebuild the guide. For students and users, the most important Makefile will be ⟨*MakefileBrew* 11a⟩. This Makefile is located under the installation root and is able to reinstall BREW.

11a ⟨*MakefileBrew* 11a⟩≡
⟨*some basic variables for Brew deployment* 12e⟩
⟨*parts of makefile for Brew deployment* 12a⟩

The ⟨*MakefileGuide* 11b⟩ is the Makefile for Brew developers. It can be used to rebuild BREW, the documentation and in the source version even the master guide.

11b ⟨*MakefileGuide* 11b⟩≡
⟨*parts of makefile to build brew* 14⟩
⟨*parts of makefile for guide creation* 15a⟩

The *Makefile* should be used with *make*. However, to call a target just type *make targetname*.

5.2 Back to the roots

Whenever you are lost in space or just stucked in your lecture you can redeploy the full BREW. However you already know the *Makefile*. To reset your BREW to default there are different options.

12a ⟨*parts of makefile for Brew deployment* 12a⟩≡ (11a)

```
redeploybrew:
        ⟨redeploy BREW 12b⟩
deletebrew:
        ⟨delete BREW at default path 12c⟩
extractbrew:
        ⟨extract BREW from default path to default installation path 12d⟩
packbrew:
        ⟨tar.gz important pathes from BREW 12f⟩
packforcopy:
        ⟨tar.gz important pathes without date 13d⟩
```

The major part for redeployment is the *redeploybrew* command. It is just a wrapper (or cycle) for other commands.

12b ⟨*redeploy BREW* 12b⟩≡ (12a)

```
make deletebrew
make extractbrew
```

Hence it is self explained, the *deletebrew* command tries to delete BREW from the standard path. There is no security or other feature to prevent data loss...

12c ⟨*delete BREW at default path* 12c⟩≡ (12a)

```
rm -rvf ${appdir}
rm -rvf ${docdir}
```

However to recreate BREW it needs to be extracted from the *brew.tar.gz* file. It also creates the Brew app directory.

12d ⟨*extract BREW from default path to default installation path* 12d⟩≡ (12a)

```
tar -xzvf ${filename}
```

For a successfully build we need some nice variables in the Makefile.

12e ⟨*some basic variables for Brew deployment* 12e⟩≡ (11a)

⟨*filename for deployment or suffix* 13c⟩
⟨*some basic directories settings* 13a⟩
⟨*additionals documentation and datesettings* 13b⟩

You can even tar BREW in current state. Be careful and watch the directories which get saved....

12f ⟨*tar.gz important pathes from BREW* 12f⟩≡ (12a)

```
tar -czvf "${actualfile}" ${allFilesToTarGz}
```

Some basic directories are explained within the variable settings

13a ⟨*some basic directories settings* 13a⟩≡ (12e)

```
appdir := app
docdir := doc
sources := ${appdir}/src
lib := ${appdir}/lib
webapps := ${appdir}/webapps
```

There are also some additional files and date settings

13b ⟨*additionals documentation and datesettings* 13b⟩≡ (12e)

```
additionals := makefile ${appdir}/.project ${appdir}/.classpath
documentation := ${docdir}/userguide.pdf ${docdir}/README
actualfile = $(shell date)_${filename}
allFilesToTarGz := ${sources} ${lib} ${webapps} ${additionals} ${documentation}
```

The basic filename which will be the main deployment file is *brew.tar.gz*. Also this is the suffix for the backup script.

13c ⟨*filename for deployment or suffix* 13c⟩≡ (12e)

```
filename := brew.tar.gz
```

For completeness you can tar.gz it without date in the filename

13d ⟨*tar.gz important pathes without date* 13d⟩≡ (12a)

```
        tar -czvf ${filename} ${allFilesToTarGz}
```

5.3 Build BREW on your own

To create BREW from this literate programming doc, we need some Makefile entries

14 ⟨*parts of makefile to build brew* 14⟩≡ (11b)

```
buildBrew:
        make SearchController.java
        make search.jsp
        make README
SearchController.java: userguide.nw
        notangle -t8 -R"SearchController.java" \
        userguide.nw > SearchController.java
search.jsp: userguide.nw
        notangle -t8 -R"search.jsp" \
        userguide.nw > search.jsp

README: userguide.nw
        notangle -t8 -R"README" \
        userguide.nw > README

upload: userguide.nw
        scp -P 3022 README muse@127.0.0.1:/home/muse/userguide
        scp -P 3022 userguide.pdf muse@127.0.0.1:/home/muse/userguide
        scp -P 3022 Makefile muse@127.0.0.1:/home/muse/userguide
        scp -P 3022 userguide.html muse@127.0.0.1:/home/muse/userguide
```

5.4 Guidebuilding

Each guide is written as literate programming. The toolset depends on make, latex, eps2pdf, noweb, and of course gcc, javac and python interpreter.

For a basic setup there is the main Makefile This Makefile has different options

15a ⟨*parts of makefile for guide creation* 15a⟩≡ (11b)

```
edituserguide:
        ⟨edit userguide 15c⟩
cycle:
        ⟨cycle all 15b⟩
userguide.tex: userguide.nw
        ⟨userguide to tex 15e⟩
userguide.html: userguide.nw
        ⟨userguide to html 16a⟩
userguide.pdf: userguide.tex
        ⟨userguide to pdf 16b⟩
Makefile: userguide.nw
        ⟨build makefile 15d⟩
copydoc:
        ⟨copy doc to dir 16c⟩
```

The most important part is the *cycle* directive. It recreates the user guide and builds the source code from this document.

15b ⟨*cycle all* 15b⟩≡ (15a)

```
make Makefile
make buildBrew
make userguide.tex
make userguide.html
make userguide.pdf
make copydoc
```

The *edituserguide* is a simple shell command. It calls vim as standard editor.

15c ⟨*edit userguide* 15c⟩≡ (15a)

```
vim userguide.nw
```

An even fancy option is to build the Makefile from this Makefile.

15d ⟨*build makefile* 15d⟩≡ (15a)

```
notangle -t8 -R"MakefileGuide" userguide.nw > MakefileGuide
notangle -t8 -R"MakefileBrew" userguide.nw > MakefileBrew
mv MakefileGuide Makefile
mv MakefileBrew ../../makefile
```

To extract the *userguide.tex* file the following directive is implemented. This will just extract the texfile, but would not compile it.

15e ⟨*userguide to tex* 15e⟩≡ (15a)

```
noweave -x -n -delay \
-latex userguide.nw > userguide.tex
```

To build a pretty looking html following directive will produce this part:

16a ⟨*userguide to html* 16a⟩≡ (15a)

```
noweave -filter l2h -index \
-html userguide.nw | htmltoc > userguide.html
```

Further the tex code needs to be compiled:

16b ⟨*userguide to pdf* 16b⟩≡ (15a)

```
pdflatex userguide.tex
#bibtex userguide.tex
pdflatex userguide.tex
pdflatex userguide.tex
```

At last we need to copy the important documents to the correct folder

16c ⟨*copy doc to dir* 16c⟩≡ (15a)

```
cp userguide.pdf ../
cp README ../
```

6 Appendix

7 Readme and stuff

17 ⟨*README* 17⟩≡

```
Welcome to BREW

####Open BREW
This is more how to start the IDE.
Hence we suppose you have no running instance from eclipse.
Use your right mouse button and click "Eclipse"
You can also start eclipse by just typing "eclipse" in xterm

####Eclipse is open....next Step?
You have successfully started eclipse.
The first time you should see this README file

When BREW is not the default project or loaded...
Open BREW

Just use File-->Open
Select the desired folder and just open it.

You should open it, no import, extract or anything else

####BREW is loaded...next Step?
Normally you can use the "Run" Button to start BREW

However when you imported BREW or did other unusal things...
and there is no "Run" Button or does not work
You have to start BREW from the main file.
The mainfile is located under src-->edu-->hm-->muse-->TomcatServer.java.


####BREW has been started...next Step?
You can call BREW with a Web Browser.
The Browser can be found by right click --> Browser
Normally the page of BREW is the landing page

Or use "localhost:8081/secu" as start page.

####I want to change sth
When you want to redeploy the source code:
Change your source code
Stop BREW (red square)
Start BREW
```

```
Your changes had been deployed
```

7.1 License and stuff

This is the general license for BREW

18a ⟨*some fancy license stuff* 18a⟩≡ (8a 18b)

```
/*
 * **
 *  *                                                __          ____
 *  *     /'\_/`\                  __        /\ \        /\  _`\
 *  *    /\      \  __  __   ___ /\_\    ___\ \ \___    \ \,\L\_\     __    ___  __  __  _
 *  *    \ \ \__\ \/\ \/\ \/' _ `\/\ \  /'___\ \  _ `\   \/_\__ \   /'__`\ /'___\\ \/\ \/\
 *  *     \ \ \_/\ \ \ \_\ \\ \/\ \ \ \/\ \__/\ \ \ \ \    /\ \L\ \/\  __//\ \__/ \ \_\ \
 *  *      \ \_\\ \_\ \____/ \_\ \_\ \_\ \____\\ \_\ \_\   \ `\____\ \____\ \____\ \____/\
 *  *       \/_/ \/_/\/___/ \/_/\/_/\/_/\/____/ \/_/\/_/    \/_____/\/____/\/____/\/___/
 *  *
 *  *
 *  *
 *  *     ____                                                 __          ____
 *  *    /\  _`\                                              /\ \        /\  _`\
 *  *    \ \ \L\ \     __    ____    __     __     _ __  ___\ \ \___    \ \ \L\_\  _ __  _
 *  *     \ \ ,  /   /'__`\ /',__\ /'__`\ /'__`\  /\`'__\'___\ \  _ `\   \ \ \L_L /\`'__\
 *  *      \ \ \\ \ /\  __//\__, `\\  __//\ \L\.\_\ \ \/\ \__/\ \ \ \ \   \ \ \/, \ \ \/\
 *  *       \ \_\ \_\ \____\/\____/ \____\ \__/.\_\\ \_\ \____\\ \_\ \_\   \ \____/\ \_\ \
 *  *        \/_/\/ /\/____/\/___/ \/____/\/__/\/_/ \/_/\/____/ \/_/\/_/    \/___/  \/_/\
 *  *
 *  *    This file is part of BREW.
 *  *
 *  *    BREW is free software: you can redistribute it and/or modify
 *  *    it under the terms of the GNU General Public License as published by
 *  *    the Free Software Foundation, either version 3 of the License, or
 *  *    (at your option) any later version.
 *  *
 *  *    BREW is distributed in the hope that it will be useful,
 *  *    but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  *    GNU General Public License for more details.
 *  *
 *  *    You should have received a copy of the GNU General Public License
 *  *    along with BREW.  If not, see <http://www.gnu.org/licenses/>.
 *
 */
```

And we want to have a simple file for this license

18b ⟨*LICENSE* 18b⟩≡

⟨*some fancy license stuff* 18a⟩

7.2 imports for the different files

19a ⟨*some import statements for the SearchController* 19a⟩≡ (8a)

```
package edu.hm.muse.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
```

19b ⟨*TomcatServer.java* 19b⟩≡

⟨*Main File* 19c⟩

19c ⟨*Main File* 19c⟩≡ (19b)

```
public class TomcatServer {
    public static void main(String[] args) throws ServletException, LifecycleException, I

        Tomcat tomcat = new Tomcat();
        tomcat.setPort(8081);
        tomcat.setBaseDir(".");
        Context ctx = tomcat.addWebapp("/secu", "secu");
        tomcat.start();
        tomcat.getServer().await();
    }

}
```