Alignak Web UI Documentation

Release 0.12.1

Frédéric MOHIER

Nov 13, 2018

Contents

CHAPTER 1

Project version:

Main version: 0.12, release: 0.12.1

CHAPTER 2

Documentation contents:

2.1 Introduction

2.1.1 About Alignak

Alignak is an open source monitoring framework written in Python under the terms of the GNU Affero General Public License . It is a fork of the Shinken project.

More information about Alignak is available in the Alignak documentation.

2.1.2 About Alignak WebUI

The Alignak Web User Interface is an open source Web application written in Python under the terms of the GNU Affero General Public License .

It is intended to be used in the Alignak project as a Web User Interface to allow users to visualize and interact with the monitoring framework.

Alignak WebUI

The main idea when developing this application is the configurability and the best suitability for the needs of the different categories of users involved around the monitored system.

Features

Alignak WebUI has a lot of features:

- nice and modern UI design, thanks to using the Bootstrap and Material Design librairies
- user role management, thanks to the best use of the Alignak backend features, the WebUI user is granted rights on the monitored objects
- configurability, many configuration parameters allow to customize the look and feel of the interface. Each user can have its own view on the system.
- external widgets, many of the WebUI views are available to be used by an external application (such as the Glpi application and its Plugin Monitoring)

Release cycle

Alignak WebUI has no strict schedule for releasing.

Feature addition is discussed throught the project issues. Each feature is discussed in a separate issue.

2.2 Installation

2.2.1 Requirements

To use this application, you first need to install some Python modules that are listed in the `requirements.txt` file:

```
# Too general libraries to specify a version
future
six

# configparser is used to parse command line
configparser; python_version == '2.7'
# docopt is used by the alignak_environment script
docopt

Bottle>=0.12.9,<0.13
Beaker==1.10.0

# Alignak WebUI supports the same version of CherryPy as Alignak
CherryPy==15.0.0

requests

# Not directly useful
# pymongo==3.7
# For application localization
python-gettext

# Colored console log
termcolor==1.1.0

python-dateutil>=2.4.2
pytz

# uWSGI server - a recommended Web server to run the application in a␣
↪production environment
# uncomment this to install from the Python repository if not installed by␣
↪the server packaging
# uwsgi==0.2.16

# Alignak backend (most recent version)
alignak-backend
# Alignak backend (most recent develop)
# -e git+git://github.com/Alignak-monitoring-contrib/alignak-backend.
↪git@develop#egg=alignak_backend

# Alignak backend client (most recent version)
alignak-backend-client
# Alignak backend client (most recent develop)
# -e git+git://github.com/Alignak-monitoring-contrib/alignak-backend-client.
↪git@develop#egg=alignak_backend_client
```

Note: if you proceed to an end-user installation with pip, the required modules are automatically installed.

uWSGI

We recommend to use uWSGI as an application server for the Alignak Web UI.

You can install uWsgi with the python packaging:

```
sudo pip install uWSGI
```

To get pip3 for Python 3 packages installation:

```
sudo apt-get install python3-pip
sudo pip3 install uWSGI
```

If you prefer using your Unix/Linux ditribution packaging to install uWSGI and the alignak Web UI, please refer to your distribution packages for installing. You will also need to install the uWSGI Python plugin.

As an example on Debian (for python 2):

```
sudo apt-get install uwsgi uwsgi-plugin-python
```

As an example on Debian (for python 3):

```
sudo apt-get install uwsgi uwsgi-plugin-python3
```

As an example on CentOS (for python 2):

```
# You need EPEL repository!
sudo yum install epel-release

sudo yum install uwsgi uwsgi-plugin-python
```

Warning: If you get some errors with the plugins, you will need to set some options in the alignak Web UI */usr/local/share/alignak-webui/etc/uwsgi.ini* configuration file or in the installed systemctl service unit. See this configuration file commented accordingly.

2.2.2 Install on Debian-like Linux

Installing Alignak Web UI for a Debian based Linux distribution (eg. Debian, Ubuntu, etc.) is using `deb` packages and it is the recommended way. You can find packages in the Alignak dedicated repositories.

To proceed with installation, you must register the alignak repository and store its public key on your system. This script is an example (for Ubuntu 16) to be adapted to your system:

Create the file */etc/apt/sources.list.d/alignak.list* with the following content:

```
# Alignak DEB stable packages
sudo echo deb https://dl.bintray.com/alignak/alignak-deb-stable xenial main | sudo␣
↪tee -a /etc/apt/sources.list.d/alignak.list
```

If your system complains about missing GPG key, you can add the publib BinTray GPG key:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv D401AB61
```

If you wish to use the non-stable versions (eg. current develop or any other specific branch), you can also add the repository source for the test versions:

```
# Alignak DEB testing packages
sudo echo deb https://dl.bintray.com/alignak/alignak-deb-testing xenial main | sudo␣
↪tee -a /etc/apt/sources.list.d/alignak.list
```

Note: According to your OS, replace {xenial} in the former script example:

- Debian 8: `jessie`
- Ubuntu 16.04: `xenial`
- Ubuntu 14.04: `trusty`
- Ubuntu 12.04: `precise`

And then update the repositories list:

```
sudo apt-get update
```

Once the download sources are set, you can simply use the standard package tool to have more information about Alignak packages and available versions:

```
apt-cache search python-alignak-webui
```

Or you can simply use the standard package tool to install Alignak Web UI:

```
sudo apt install python-alignak-webui

# Check Alignak Web UI installation
# It copied the default shipped files and sample configuration.
ll /usr/local/share/alignak-webui/

# It installed the Alignak systemd services
ll /lib/systemd/system/alignak*
   -rw-r--r-- 1 root root 1715 juil.  1 11:12 /lib/systemd/system/alignak-uwsgi.
↪service

# Alignak Web UI service status
sudo systemctl status alignak-webui
$ sudo systemctl status alignak-webui
    alignak-webui.service - uWSGI instance to serve Alignak Web UI
      Loaded: loaded (/lib/systemd/system/alignak-webui.service; enabled; vendor␣
↪preset: enabled)
      Active: inactive (dead)
```

Note: that immediately after the installation the *alignak-webui* service is enabled and started! This is a side effect of the packaging tool that is used (*fpm*).

A post-installation script (repository *bin/python-post-install.sh*) is started at the end of the installation procedure to install the required Python packages. This script is copied during the installation in the default installation directory: */usr/local/share/alignak-webui*. It is using the Python pip tool to get the Python packages listed in the default installation directory *requirements.txt* file.

Note: this hack is necessary to be sure that we use the expected versions of the needed Python libraries...

It is recommended to set-up a log rotation because the Alignak backend log may be really verbose! Using the `logrotate` is easy. A default file is shipped with the installation script and copied to the */etc/logrotate.d/alignak-backend* with this content:

```
"/var/log/alignak-webui/*.log" {
  copytruncate
  daily
  rotate 5
  compress
  delaycompress
  missingok
  notifempty
}
```

A log rotation file for uWsgi is also shipped with the installation script and copied to the */etc/logrotate.d/uwsgi* with this content:

```
"/var/log/uwsgi/alignak-webui.log" {
  copytruncate
  daily
  rotate 5
  compress
  delaycompress
  missingok
  notifempty
}
```

Note: for Python 3 version, replace `python` with `python3` in the package and post-installation script names.

2.2.3 Install on RHEL-like Linux

Installing Alignak Web UI for an RPM based Linux distribution (eg. RHEL, CentOS, etc.) is using `rpm` packages and it is the recommended way. You can find packages in the Alignak dedicated repositories.

To proceed with installation, you must register the alignak repositories on your system.

Create the file */etc/yum.repos.d/alignak-stable.repo* with the following content:

```
[Alignak-rpm-stable]
name=Alignak RPM stable packages
baseurl=https://dl.bintray.com/alignak/alignak-rpm-stable
gpgcheck=0
repo_gpgcheck=0
enabled=1
```

And then update the repositories list:

```
sudo yum repolist
```

If you wish to use the non-stable versions (eg. current develop or any other specific branch), you can also create a repository source for the test versions. Then create a file */etc/yum.repos.d/alignak-testing.repo* with the following content:

```
[Alignak-rpm-testing]
name=Alignak RPM testing packages
baseurl=https://dl.bintray.com/alignak/alignak-rpm-testing
gpgcheck=0
repo_gpgcheck=0
enabled=1
```

The Alignak packages repositories contain several version of the application. The versioning scheme is the same as the Alignak one.

Once the download sources are set, you can simply use the standard package tool to have more information about Alignak pack

```
yum search alignak-webui
     Loaded plugins: fastestmirror
     Loading mirror speeds from cached hostfile
     * base: mirrors.atosworldline.com
     * epel: mirror.speedpartner.de
     * extras: mirrors.atosworldline.com
     * updates: mirrors.standaloneinstaller.com
     ================================================================================␣
↪N/S matched: alignak␣
↪==============================================================================
     ...
     ...
     alignak-webui.noarch : Alignak WebUI, Web User Interface for Alignak

yum info python-alignak-webui
     Modules complémentaires chargés : fastestmirror
     Loading mirror speeds from cached hostfile
     * base: ftp.rezopole.net
     * epel: mirror.miletic.net
     * extras: mirror.plusserver.com
     * updates: ftp.rezopole.net
     Paquets installés
     Nom                 : alignak-webui
     Architecture        : noarch
     Version             : packaging
     Révision            : 1
     Taille              : 12 M
     Dépôt               : installed
     Depuis le dépôt     : Alignak-rpm-testing
     Résumé              : Alignak WebUI, Web User Interface for Alignak
     URL                 : http://alignak.net
     Licence             : AGPL
     Description         : Alignak WebUI, Web User Interface for Alignak
```

Or you can simply use the standard package tool to install Alignak Web UI and its dependencies.

```
sudo yum install python-alignak-webui

# Check Alignak Web UI installation
# It copied the default shipped files and sample configuration.
ll /usr/local/share/alignak-webui/
   -rw-rw-r--. 1 root root  527 10 juil. 21:03 requirements.txt
```

A post-installation script (repository *bin/python-post-install.sh*) must be executed at the end of the installation procedure to install the required Python packages. This script is copied during the installation in the default installation

directory: */usr/local/share/alignak-webui*. It is using the Python pip tool to get the Python packages listed in the default installation directory *requirements.txt* file.

```
sudo /usr/local/share/alignak-webui/python-post-install.sh
```

Note: this hack is necessary to be sure that we use the expected versions of the needed Python libraries...

It is recommended to set-up a log rotation because the Alignak Web UI log may be really verbose! Using the `logrotate` is easy. A default file is shipped with the installation script and copied to the */etc/logrotate.d/alignak-webui* with this content:

```
"/var/log/alignak-webui/*.log" {
  copytruncate
  daily
  rotate 5
  compress
  delaycompress
  missingok
  notifempty
}
```

A log rotation file for uWsgi is also shipped with the installation script and copied to the */etc/logrotate.d/alignak-webui-uwsgi* with this content:

```
"/var/log/uwsgi/alignak-backend.log" {
  copytruncate
  daily
  rotate 5
  compress
  delaycompress
  missingok
  notifempty
}
```

To terminate the installation of the system services you must:

```
# For Python 2 installation
sudo cp /usr/local/share/alignak-webui/bin/systemd/python2/alignak-webui-centos7.
↪service /etc/systemd/system/alignak-webui.service

# For Python 3 installation
sudo cp /usr/local/share/alignak-webui/bin/systemd/python3/alignak-webui-centos7.
↪service /etc/systemd/system/alignak-webui.service

ll /etc/systemd/system
   -rw-r--r--. 1 root root  777 May 24 17:48 /lib/systemd/system/alignak-webui.service

sudo systemctl enable alignak-webui
   Created symlink from /etc/systemd/system/multi-user.target.wants/alignak-webui.
↪service to /usr/lib/systemd/system/alignak-webui.service.
```

Note: for Python 3 version, replace `python` with `python3` in the package and post-installation script names.

2.2.4 Installation with PIP

Note that the recommended way for installing on a production server is mostly often to use the packages existing for your distribution.

Nevertheless, the pip installation provides: - a startup script using an uwsgi server, - for FreeBSD users, an rc.d service script, - for systemctl based systems (Debian, CentOS), an alignak-webui service unit.

All this stuff is available in the repository *bin* directory and is copied locally in the */usr/local/share/alignak-webui* directory.

End user installation

Installing with pip:

```
sudo pip install alignak-webui
```

The required Python modules are automatically installed if not they are not yet present on your system.

From source

Installing from source:

```
git clone https://github.com/Alignak-monitoring/alignak-webui
cd alignak-webui
pip install .
```

For contributors

If you want to hack into the codebase (e.g for future contribution), just install like this:

```
pip install -e .
```

2.2.5 Install from source without pip

If you are on Debian:

```
sudo apt-get -y install python python-dev python-pip git
```

Get the project sources:

```
git clone https://github.com/Alignak-monitoring/alignak-webui
```

And then install:

```
cd alignak-webui
pip install .
```

2.3 Configuration

The application runs without any extra configuration file with its default parameters. Nevertheless, the application is best used when suited to user's neeeds ;)

2.3.1 Application log

The application uses the Python logger facility to produce an activity log. The log can be configured thanks to a configuration file `logging.json` located on your system in a directory according to whether your system is a Linux (Debian) or Unix (FreeBSD) distribution. This location is, most often, */usr/local/etc/alignak-webui/*.

Thus, the application searches in several location for a logger configuration file:

- /usr/local/etc/alignak-webui/logging.json
- /etc/alignak-webui/logging.json
- ~/alignak-webui/logging.json
- ./etc/logging.json
- ./alignak-webui/etc/logging.json
- ./logging.json

The first file found is retained as the application logger configuration file.

If an environment variable `ALIGNAK_WEBUI_LOGGER_FILE` exists, this variable is used by the application as the only configuration file name to be loaded by the application. It allows to **override the default file list**.

The application stores its log file(s) in a directory searched among:

- /usr/local/var/log/alignak-webui
- /var/log/alignak-webui
- /tmp/alignak-webui

The log file(s) location is the first found directory. If an environment variable `ALIGNAK_WEBUI_LOG_DIR` exists, this variable is used by the application as the log directory. It allows to **override the default log location**.

The default logging behavior is to log to the console and in a daily rotating file at INFO log level with a local time date.

Some simple modifications:

- to change log level: change the root logger *level* to DEBUG, WARNING, ...
- to set date time as UTC: change the console handler formatter to *utc*

Else, if you are aware of the Python logger configuration, update the file according to your needs:

```
{
  "version": 1,
  "disable_existing_loggers": false,
  "formatters": {
    "utc": {
      "()": "alignak_webui.utils.logger.UTCFormatter",
      "format": "[%(asctime)s] %(levelname)s: [%(name)s] %(message)s"
    },
    "local": {
      "format": "[%(asctime)s] %(levelname)s: [%(name)s] %(message)s"
    }
  },

  "handlers": {
    "console": {
      "class": "alignak_webui.utils.logger.ColorStreamHandler",
      "level": "DEBUG",
```

(continues on next page)

(continued from previous page)

```
      "formatter": "local",
      "stream": "ext://sys.stdout"
    },
    "file": {
      "class": "logging.handlers.TimedRotatingFileHandler",
      "level": "DEBUG",
      "formatter": "local",
      "filename": "alignak-webui.log",
      "when": "midnight",
      "interval": 1,
      "backupCount": 7,
      "encoding": "utf8"
    }
  },

  "root": {
    "level": "INFO",
    "handlers": ["console", "file"]
  }
}
```

Note that the Web UI is intended to be launched as an uWSGI application and that uWSGI will be configured to log the application console output to a file...

2.3.2 Application configuration

Configuration file location

The application can be configured thanks to a configuration file. When installed for an end user, the configuration file `settings.cfg` is located on your system in a directory according to whether your system is a Linux (Debian) or Unix (FreeBSD) distribution. This location is determined by the Python setup.py script.

Thus, the application searches in several location for a configuration file:

- /usr/local/etc/alignak-webui/settings.cfg
- /etc/alignak-webui/settings.cfg
- ~/alignak-webui/settings.cfg
- ./etc/settings.cfg
- ./alignak-webui/etc/settings.cfg
- ./settings.cfg

Each file found takes precedence over the previous files. As of it, for the same parameter with different values in */usr/local/etc/alignak-webui/settings.cfg* and *./settings.cfg*, the retained value will be the one configured in *./settings.cfg*.

If an environment variable `ALIGNAK_WEBUI_CONFIGURATION_FILE` exists, this variable is used by the application as the only configuration file name to be loaded by the application. It allows to **override the default file list**.

If an environment variable *ALIGNAK_WEBUI_UWSGI_FILE* exist, the *alignak-webui-uwsgi* script will use the file name defined in this variable as the uWSGI configuration file.

If an environment variable `ALIGNAK_WEBUI_CONFIGURATION_THREAD` exists, the application will check periodically if its configuration file changed. If the configuration file modification time changed, the configuration is reloaded by the application.

If an environment variable `BOTTLE_DEBUG` exists and is set to '1', the Bottle application server will run in debug mode. If an environment variable `ALIGNAK_WEBUI_DEBUG` exists and is set to '1', the application will run in debug mode; which means that the application logs will be set to a DEBUG level.

If an environment variable `ALIGNAK_WEBUI_BACKEND` exists, the value of this variable will override the one defined in the configuration file (`alignak_backend`).

If an environment variable `ALIGNAK_WEBUI_WS` exists, the value of this variable will override the one defined in the configuration file (`alignak_ws`).

Configuration file format

This file is a text file in classic `.ini` file format. It is parsed using Python ConfigParser module.

Sections are introduced by a [section] header, and contain name = value entries.

Lines beginning with # or ; are ignored as comments.

Strings don't need quotes.

Multi-valued strings can be created by indenting values on multiple lines.

Boolean values can be specified as on, off, true, false, 1, or 0 and are case-insensitive.

Environment variables can be substituted in by using dollar signs: $WORD ${WORD} will be replaced with the value of WORD in the environment. A dollar sign can be inserted with $$. Missing environment variables will result in empty strings with no error.

A percent sign can be inserted with %%.

Configuration parameters

Note: The default configuration file contains a commented copy of all the available parameters.

Note: please do not change these parameters unless you know what you're doing!

[bottle] section

This section contains parameters to configure the base Web server.

- **host**, interface the application listens to (default: *127.0.0.1*)
- **port**, TCP port the application listens to (default: *8868*)
- **debug**, to make the server run in debug mode (only useful for developers)

With the default configuration the application server listens on TCP port 5001 of all interfaces.

[session] section

This section contains parameters to configure the application user's sessions. Thanks to those parameters it is possible to adapt the session duration according to your needs. This requires to be aware of the Web client / server session handling to make some modifications in this section.

As a default, the user session is valid from the login time up to the client's browser closing, allowing to have infinite sessions to use the Web UI on stand-alone monitors;)

[Alignak-WebUI] section

This section contains parameters to configure the application.

- **alignak_backend**, Alignak backend endpoint (default: *http://127.0.0.1:5000*)
- **alignak_ws**, Alignak Web Services endpoint (default: *http://127.0.0.1:8888*)
- **debug**, to make the application run in debug mode (much more log in the log file!)
- **about_name**, application name in About modal box (default is defined in source code)
- **about_version**, application name in About modal box (default is defined in source code)
- **about_copyright**, application copyright in About modal box (default is defined in source code)
- **about_release**, application release notes in About modal box (default is defined in source code)
- **login_text**, welcome text on the login form (default: *Welcome!
 Log-in to use the application*)
- **company_log**, logo image used on the login form (default: *'/static/images/default_company.png'*)
- **webui_logo**, logo image used in the application footer (default: */static/images/logo_webui_xxs.png*)
- **play_sound**, plays a sound when a new problem is raised (default: *no*)
- **refresh_period**, page refresh period in seconds (default: *60*). Use 0 to disable page refresh.

- **header_refresh_period**, page header refresh period in seconds (default: *30*). Use 0 to disable page header refresh.
- **locale**, language file to use (default: *en_US*). Language files are located in *locales* sub-directory.
- **timezone**, preferred timezone for dates (default: *Europe/Paris*).
- **timeformat**, default date format (default: *%Y-%m-%d %H:%M:%S*).
- **cors_acao**, CORS Access Control Allow Origin for external application access (default: *127.0.0.1*).
- **grafana**, Grafana application URL (default: empty value). When this parameter is present, the WebUI will try to display Grafana panels for the hosts/services if a panel definition exists in the data fetched from the Alignak Backend.
- **livestate_layout**, configure the layout to be used in the livestate view: single table, multiple panels or tabbed view, for each business impact level

2.3.3 UI rendering

Alignak WebUI is highly configurable and even some UI rendering can be configured to best suit the user's needs. This is what is introduced in the next chapters; each one made for a specific section of the Alignak WebUI configuration file.

[Alignak-WebUI] section

- **livestate_layout**, configure the layout to be used in the livestate view: single table, multiple panels or tabbed view, for each business impact level

Some examples:

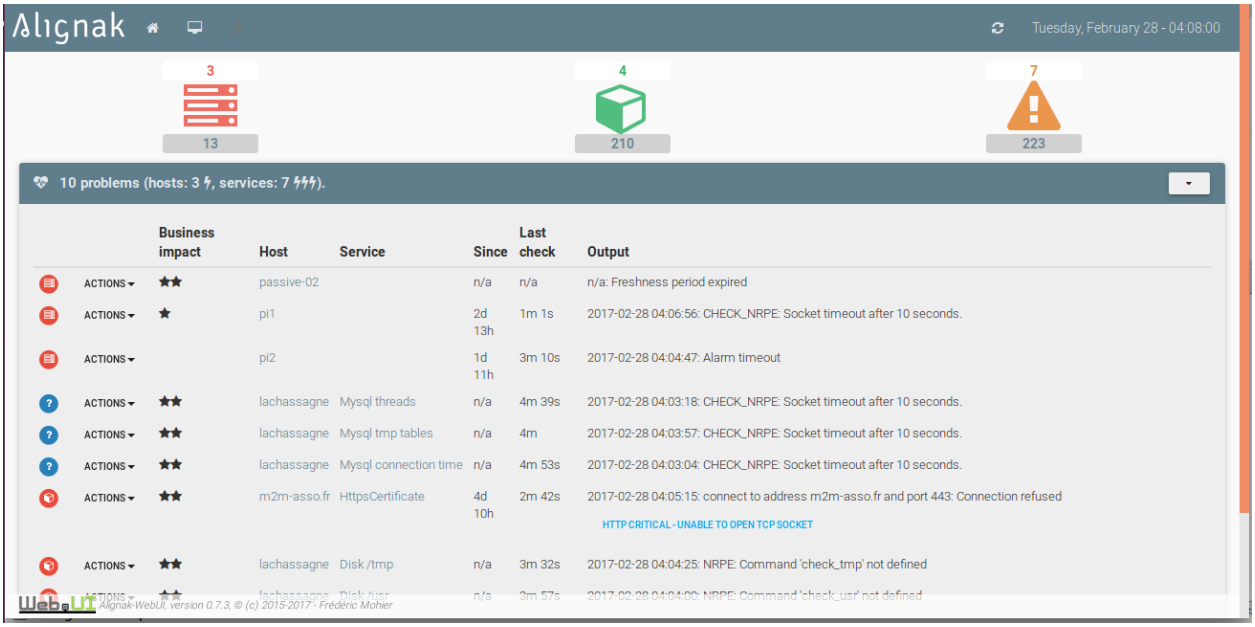


Fig. 1: Default configuration

[on_off]

This section allows to configure how the on/off (eg. enabled/disabled) is represented in the Web UI.

```
[on_off]
; Global element to be included in the HTML and including the items and the text
on=<span title="##title##" class="fa fa-fw fa-check text-success">##message##</span>

; Element to be included for each BI count
off=<span title="##title##" class="fa fa-fw fa-close text-danger">##message##</span>
```

[business_impact]

This section allows to configure how the business impact of an element is represented in the Web UI.

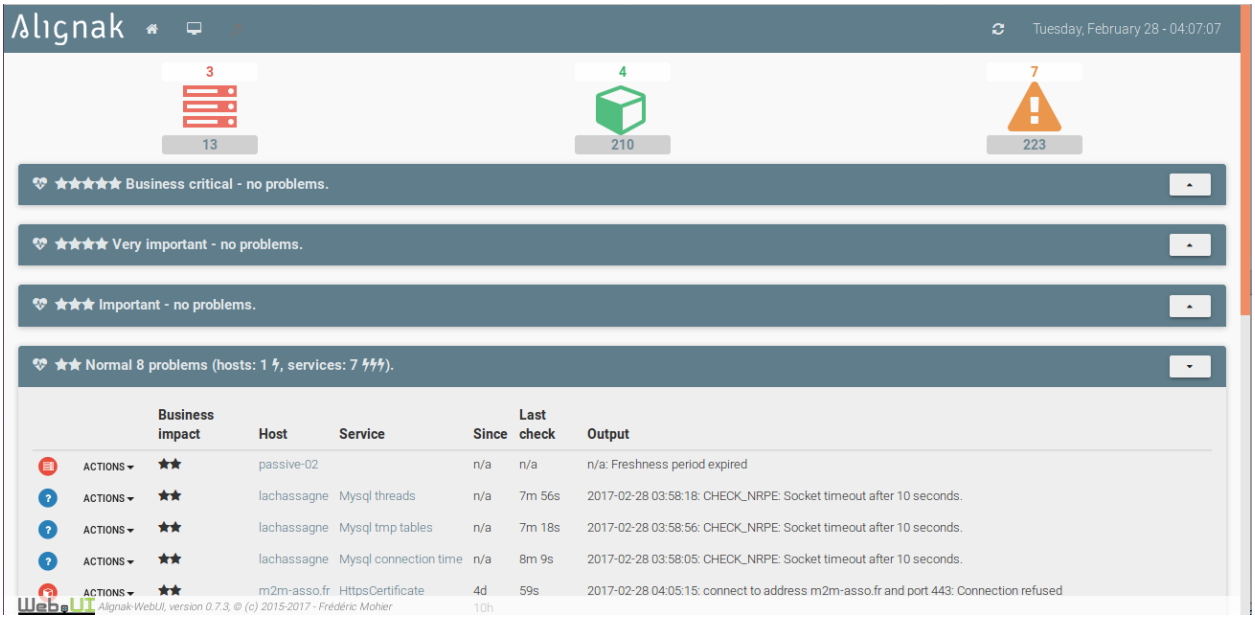


Fig. 2: Panels layout

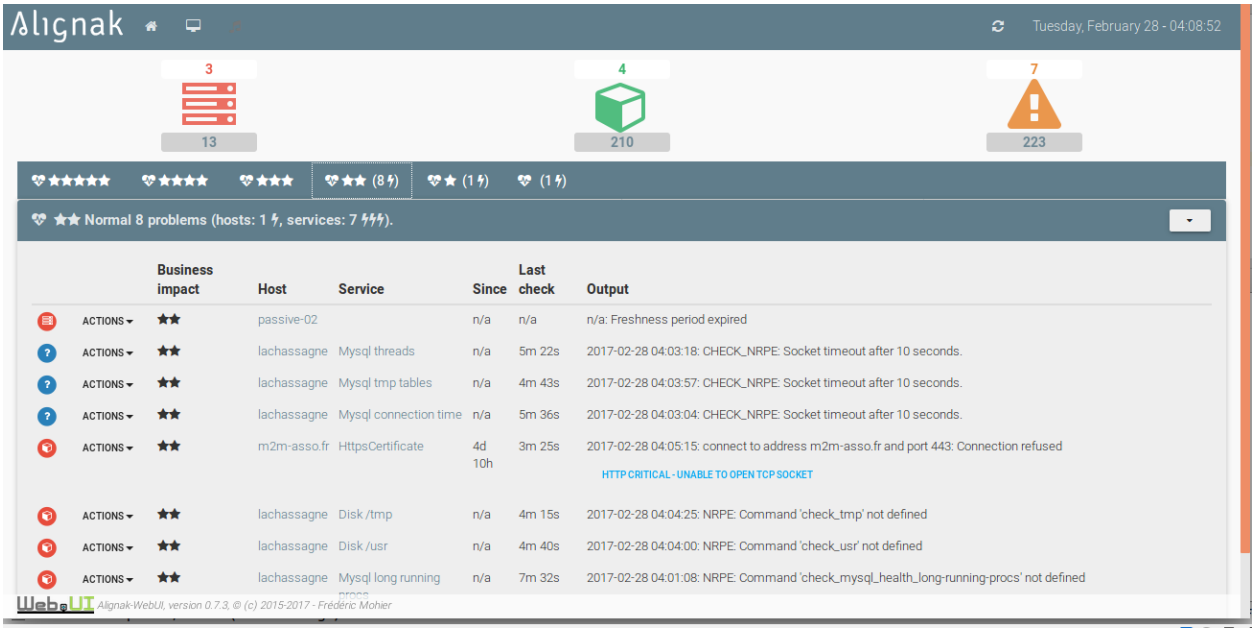


Fig. 3: Tabs

```
[business_impact]
; Global element to be included in the HTML and including the items and the text
;global=<span class="text-default">##items##</span><span> ##text##</span>

; Element to be included for each BI count
;item=<span class="fa fa-trophy"></span>
; If item is empty, then the following unique is used in place
;item=

; Unique element
; ##bi## will be replaced with the business impact level value
;unique=<div style="display: inline; font-size:0.9em;" title="##text##"><span class=
↪"fa-stack"><span class="fa fa-circle fa-stack-2x"></span><span class="fa fa-stack-
↪1x fa-inverse">##bi##</span></span></div>

; Number of elements to remove from the real business impact
; 0 is meaning that the defined item will be repeated twice for BI=2, third for BI=3
; 2 is meaning that the defined item will not be repeated for BI=2, and once for BI=3
;less=0
```

Some examples:

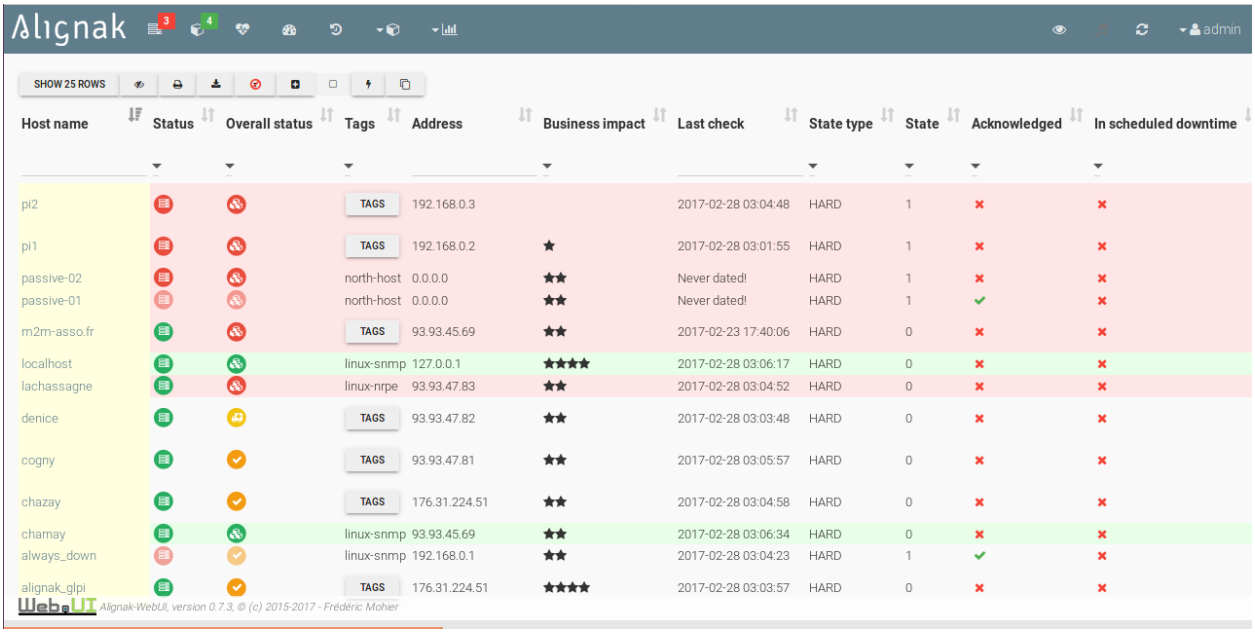


Fig. 4: Default configuration

[buttons]

This section defines patterns used by the application to build the buttons commands toolbar.

```
[buttons]
; First solution: a buttons group
; Global element to be included in the HTML
;livestate_commands=<div class="btn-group btn-group-xs btn-group-raised" role="group"␣
↪data-type="actions" title="##title##">##commands##</div>
```

(continues on next page)

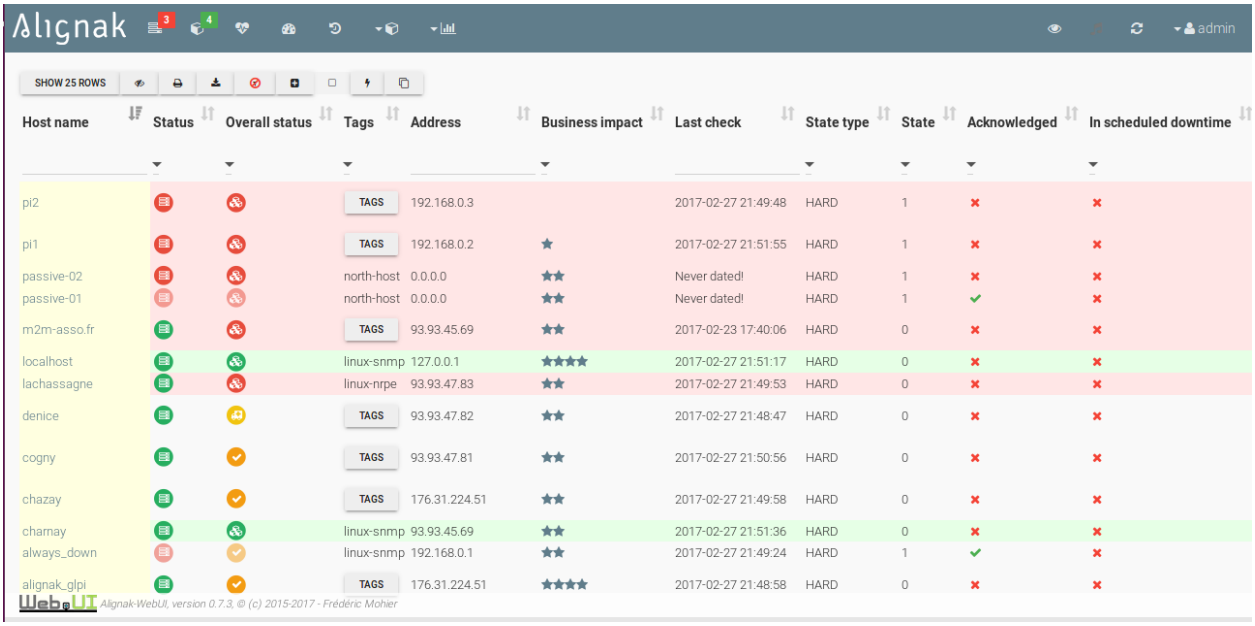


Fig. 5: Changed color

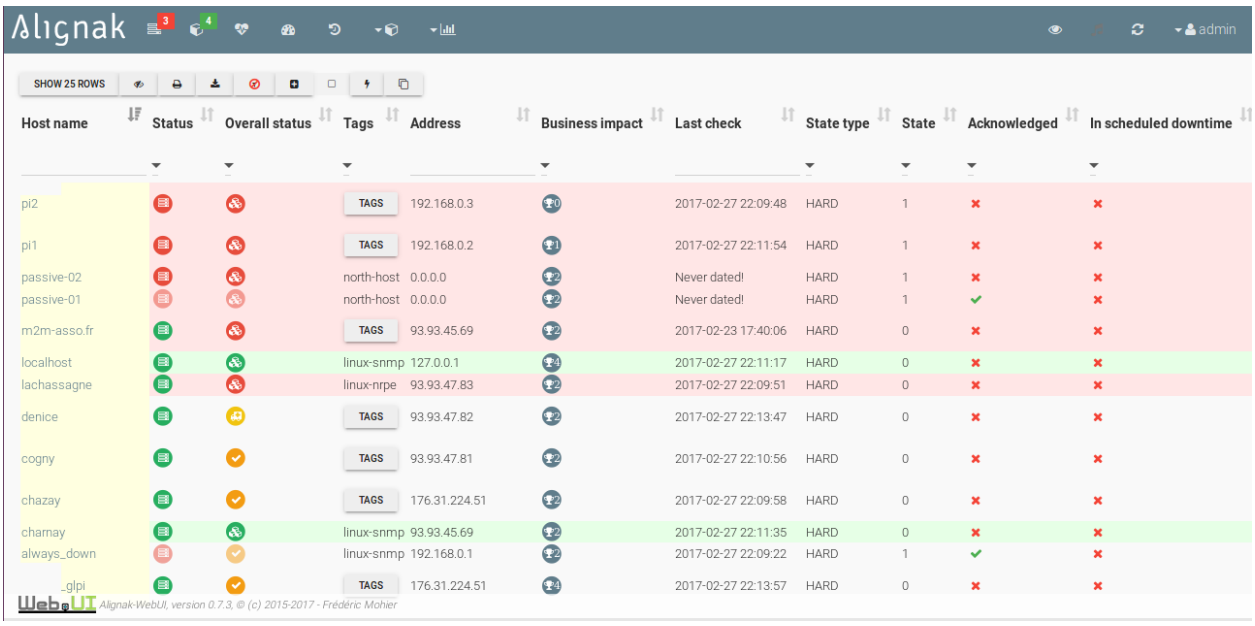


Fig. 6: Icon and text

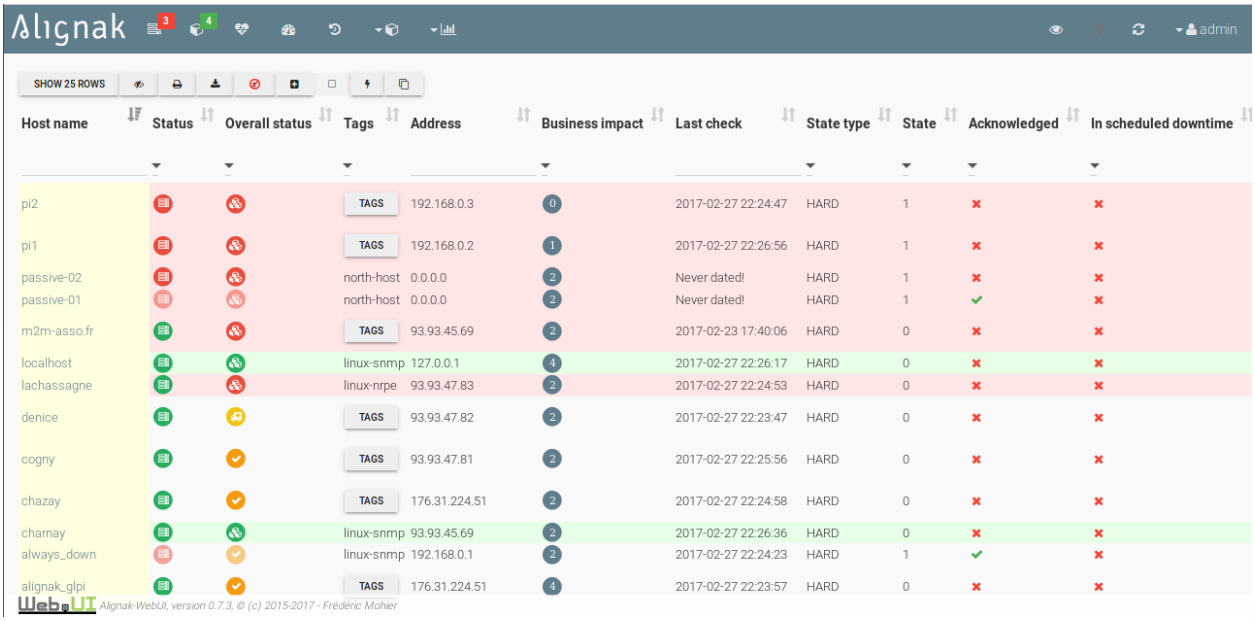


Fig. 7: Text only

(continued from previous page)

```
; Each command element to be included in the HTML
;livestate_command=<button class="btn btn-default" data-type="action" data-action="#
↪#action##" data-toggle="tooltip" data-placement="top" title="##title##" data-
↪element_type="##type##" data-name="##name##" data-element="##id##" ##disabled##><i␣
↪class="fa fa-##icon##"></i></button>

; Second solution (preferred one): a buttons dropdown list
; Global element to be included in the HTML
livestate_commands=<div class="btn-group btn-group-xs" role="group" data-type="actions
↪" title="##title##"><button type="button" class="btn btn-default dropdown-toggle"␣
↪data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">##title## <span␣
↪class="caret"></span></button><ul class="dropdown-menu">##commands##</ul></div>
; Each command element to be included in the HTML
livestate_command=<li><button class="btn btn-default" data-type="action" data-action="
↪##action##" data-toggle="tooltip" data-placement="top" title="##title##" data-
↪element_type="##type##" data-name="##name##" data-element="##id##" ##disabled##><i␣
↪class="fa fa-##icon##"></i>  ##title##</button></li>
```

[tables.lists]

This section defines patterns used by the application to build the elements lists in the tables.

```
[tables.lists]
; Button to display the list
button=<button class="btn btn-xs btn-raised" data-toggle="collapse" data-target="
↪#list_##type##_##id##" aria-expanded="false">##title##</button><div class="collapse
↪" id="list_##type##_##id##">##content##</div>

; Global element to be included in the HTML for the list
list=<ul class="list-group">##content##</ul>
```

(continues on next page)

(continued from previous page)

```
; Each command element to be included in the HTML list
item=<li class="list-group-item"><span class="fa fa-check"> ##content##</span></
↪li>

; Unique element to be included in the HTML list if the list contains only one element
unique=##content##
```

[currently]

This section defines patterns used by the application to build the currently view.

```
; Hosts states to include in the history graph
; States can be: up,down,unreachable,acknowledged,in_downtime
; Defaults to: up,down,unreachable,acknowledged,in_downtime
;hh_states=up,down,unreachable,acknowledged,in_downtime

; Hosts states history graph height
; Defaults to: 300
;hh_height=300

; Services states to include in the history graph
; States can be: ok,warning,critical,unknown,unreachable,acknowledged,in_downtime
; Defaults to: ok,warning,critical,unknown,acknowledged,in_downtime
;sh_states=ok,warning,critical,unknown,acknowledged,in_downtime

; Services states history graph height
; Defaults to: 300
;sh_height=300

; Hosts panel definition
hosts_panel=<div id="panel_hosts" class="panel panel-default">
            <div class="panel-heading clearfix">
              <strong>
                <span class="fa fa-server"></span>
                <span class="hosts-all text-white" data-count="##nb_elts##" data-
↪problems="##nb_problems##">
                      ##nb_elts## hosts ##problems##
                </span>
              </strong>

              <div class="pull-right">
                 <a href="#p_ph" class="btn btn-xs btn-raised" data-toggle="collapse">
                    <i class="fa fa-fw %%s"></i>
                 </a>
              </div>
            </div>
            <div id="p_ph" class="panel-collapse collapse %%s">
              <div class="panel-body">
              ##hosts_counters##
              <hr>
              ##hosts_percentage##
              </div>
            </div>
        </div>
```

(continues on next page)

(continued from previous page)

```
hosts_counters=
    <div class="row">
        <div class="col-xs-12 col-sm-9 text-center">
          <div class="col-xs-4 text-center">
            <a href="##hosts_table_url##?search=ls_state:UP"
              class="item_host_up" title="Up">
              <span class="hosts-count">##nb_up##</span>
            </a>
          </div>
          <div class="col-xs-4 text-center">
            <a href="##hosts_table_url##?search=ls_state:DOWN"
              class="item_host_down" title="Down">
              <span class="hosts-count">##nb_down##</span>
            </a>
          </div>
          <div class="col-xs-4 text-center">
            <a href="##hosts_table_url##?search=ls_state:UNREACHABLE"
              class="item_host_unreachable" title="Unreachable">
              <span class="hosts-count">##nb_unreachable##</span>
            </a>
          </div>
        </div>

        <div class="col-xs-12 col-sm-3 text-center">
          <a href="##hosts_table_url##?search=ls_state:acknowledged"
            class="item_host_acknowledged" title="Acknowledged">
            <span class="hosts-count">##nb_acknowledged##</span>
          </a>
          <span>/</span>
          <a href="##hosts_table_url##?search=ls_state:IN_DOWNTIME"
            class="item_host_in_downtime" title="In downtime">
            <span class="hosts-count">##nb_in_downtime##</span>
          </a>
        </div>
    </div>

hosts_percentage=
    <div class="row">
      <div class="col-xs-3 col-sm-3 text-center">
        <div class="col-xs-12 text-center">
          <a href="##hosts_table_url##" class="sla_hosts_##font##">
            <div>##pct_sla##%%</div>
            <i class="fa fa-4x fa-server"></i>
          </a>
        </div>
      </div>

      <div class="col-xs-9 col-sm-9 text-center">
        <div class="row">
          <div class="col-xs-4 text-center">
            <a href="##hosts_table_url##?search=ls_state:UP"
              class="item_host_up" title="Up">
              <span class="hosts-count">##pct_up##%%</span>
            </a>
          </div>
          <div class="col-xs-4 text-center">
```

(continues on next page)

(continued from previous page)

```
            <a href="##hosts_table_url##?search=ls_state:DOWN"
              class="item_host_down" title="Down">
              <span class="hosts-count">##pct_down##%%</span>
            </a>
          </div>
          <div class="col-xs-4 text-center">
            <a href="##hosts_table_url##?search=ls_state:UNREACHABLE"
              class="item_host_unreachable" title="Unreachable">
              <span class="hosts-count">##pct_unreachable##%%</span>
            </a>
          </div>
          <div class="col-xs-4 text-center">
            <a href="##hosts_table_url##" title="Fake">
              <span class="hosts-count"> </span>
            </a>
          </div>
          <div class="col-xs-4 text-center">
            <a href="##hosts_table_url## title="Fake">
              <span class="hosts-count"> </span>
            </a>
          </div>
        </div>

        <div class="row">
          <div class="col-xs-12 text-center">
            <a href="##hosts_table_url##?search=ls_state:acknowledged"
              class="item_host_acknowledged" title="Acknowledged">
              <span class="hosts-count">##pct_acknowledged##%%</span>
            </a>
            <span>/</span>
            <a href="##hosts_table_url##?search=ls_state:IN_DOWNTIME"
              class="item_host_in_downtime" title="In downtime">
              <span class="hosts-count">##pct_in_downtime##%%</span>
            </a>
          </div>
        </div>
      </div>
    </div>

; Services panel definition
services_panel=
    <div id="panel_services" class="panel panel-default">
        <div class="panel-heading clearfix">
          <strong>
            <span class="fa fa-cube"></span>
            <span class="services-all text-default"
                    data-count="##nb_elts##"
                    data-problems="##nb_problems##">
                  ##nb_elts## services ##problems##
            </span>
          </strong>

          <div class="pull-right">
              <a href="#p_ps" class="btn btn-xs btn-raised" data-toggle="collapse">
                  <i class="fa fa-fw %%s"></i>
              </a>
          </div>
```

(continues on next page)

(continued from previous page)

```
        </div>
        <div id="p_ps" class="panel-collapse collapse %%s">
          <div class="panel-body">
            ##services_counters##
            <hr>
            ##services_percentage##
          </div>
        </div>
    </div>

services_counters=
    <div class="row">
        <div class="col-xs-12 col-sm-9 text-center">
          <div class="col-xs-2 text-center">
            <a href="##services_table_url##?search=ls_state:OK"
              class="item_service_ok" title="Ok">
              <span class="services-count">##nb_ok##</span>
            </a>
          </div>
          <div class="col-xs-2 text-center">
            <a href="##services_table_url##?search=ls_state:WARNING"
              class="item_service_critical" title="Warning">
              <span class="services-count">##nb_warning##</span>
            </a>
          </div>
          <div class="col-xs-2 text-center">
            <a href="##services_table_url##?search=ls_state:CRITICAL"
              class="item_service_critical" title="Critical">
              <span class="services-count">##nb_critical##</span>
            </a>
          </div>
          <div class="col-xs-2 text-center">
            <a href="##services_table_url##?search=ls_state:UNKNOWN"
              class="item_service_unknown" title="Unknown">
              <span class="services-count">##nb_unknown##</span>
            </a>
          </div>
          <div class="col-xs-2 text-center">
            <a href="##services_table_url##?search=ls_state:UNREACHABLE"
              class="item_service_unreachable" title="Unreachable">
              <span class="services-count">##nb_unreachable##</span>
            </a>
          </div>
        </div>

        <div class="col-xs-12 col-sm-3 text-center">
          <a href="##services_table_url##?search=ls_state:acknowledged"
            class="item_service_acknowledged" title="Acknowledged">
            <span class="services-count">##nb_acknowledged##</span>
          </a>
          <span>/</span>
          <a href="##services_table_url##?search=ls_state:IN_DOWNTIME"
            class="item_service_in_downtime" title="In downtime">
            <span class="services-count">##nb_in_downtime##</span>
          </a>
        </div>
    </div>
```

(continues on next page)

(continued from previous page)

```
services_percentage=
    <div class="row">
        <div class="col-xs-3 col-sm-3 text-center">
            <div class="col-xs-12 text-center">
              <a href="##services_table_url##" class="sla_services_##font##">
                <div>##pct_ok###%</div>
                <i class="fa fa-4x fa-cube"></i>
              </a>
            </div>
            </div>

            <div class="col-xs-9 col-sm-9 text-center">
            <div class="row">
              <div class="col-xs-4 text-center">
                <a href="##services_table_url##?search=ls_state:OK"
                  class="item_service_ok" title="ok">
                  <span class="services-count">##pct_ok###%</span>
                </a>
              </div>
              <div class="col-xs-4 text-center">
                <a href="##services_table_url##?search=ls_state:WARNING"
                  class="item_service_warning" title="warning">
                  <span class="services-count">##pct_warning###%</span>
                </a>
              </div>
              <div class="col-xs-4 text-center">
                <a href="##services_table_url##?search=ls_state:CRITICAL"
                  class="item_service_critical" title="critical">
                  <span class="services-count">##pct_critical###%</span>
                </a>
              </div>
              <div class="col-xs-4 text-center">
                <a href="##services_table_url##?search=ls_state:UNKNONW"
                  class="item_service_unknown" title="unknown">
                  <span class="services-count">##pct_unknown###%</span>
                </a>
              </div>
              <div class="col-xs-4 text-center">
                <a href="##services_table_url##?search=ls_state:UNREACHABLE"
                  class="item_service_unreachable" title="unreachable">
                  <span class="services-count">##pct_unreachable###%</span>
                </a>
              </div>
            </div>

            <div class="row">
              <div class="col-xs-12 text-center">
                <a href="##services_table_url##?search=ls_state:ACKNOWLEDGED"
                  class="item_service_acknowledged" title="acknowledged">
                  <span class="services-count">##pct_acknowledged###%</span>
                </a>
                <span>/</span>
                <a href="##services_table_url##?search=ls_state:IN_DOWNTIME"
                  class="item_service_in_downtime" title="in_downtime">
                  <span class="services-count">##pct_in_downtime###%</span>
```

(continues on next page)

(continued from previous page)

```
                </a>
              </div>
            </div>
        </div>
    </div>
```

[items] section

This section defines patterns used by the application to build the elements icons. **TO BE COMPLETED**

2.4 Run

2.4.1 Maintenance

The WebUI application uses the Beaker session middleware and it stores the user sessions in files in the */tmp/Alignak-WebUI/sessions* directory. You will need to remove the oldest files in this directory:

```
find /tmp/Alignak-WebUI/sessions -mtime +3 -exec rm {} \;
```

Note that most often the */tmp* directory is cleant on a regular basis and nothing special is to be done;)

When using the WebUI with the provided script, some log files are stored in the */usr/local/var/log/alignak-webui* (or */var/log/alignak-webui*). Those files should also be handled else their size will grow indefinitely...

2.4.2 Production mode

The alignak WebUI installation script used when you install with pip:

- creates a *alignak-webui-uwsgi* launch script located in */usr/local/bin*
- stores the *uwsgi.ini* configuration file in */usr/local/etc/alignak-webui*

Thanks to this, you can simply run:

```
alignak-webui-uwsgi
```

The `alignak-webui-uwsgi` script can receive a parameter for the configuration file to use. As default, it will use the */usr/local/etc/alignak-webui/uwsgi.ini* file.

The Alignak Web UI logs its activity in two files that are located in */usr/local/var/log/alignak-webui*:

- *webui-access.log* contains all the API HTTP requests
- *webui-error.log* contains the other messages: start, stop, activity log, ...

Warning: If you do not have those files when the WebUI is started, make sure that the user account used to run the backend is allowed to write in the */usr/local/var/log/alignak-webui* directory ;)

To stop / reload the Alignak WebUI application:

```
# Ctrl+C in the session where you started the alignak-webui-uwsgi script will stop␣
↪the WebUI

# To gracefully reload all the workers
$ kill -SIGHUP `cat /tmp/alignak-webui.pid`

# To gently kill all the workers
$ kill -SIGTERM `cat /tmp/alignak-webui.pid`

# To brutally kill all the workers
$ kill -SIGINT `cat /tmp/alignak-webui.pid`
```

2.4.3 System service mode

The repository *bin* directory includes some examples:

- an rc.d sample script for BSD systems
- an alignak-webui service unit example for systemd based systems (Debian, CentOS)

Thanks to this, you can edit and easily update the sample scripts to suit your system configuration. Environment variables defined in the service file allow to easily configure the application.

Then, you can:

```
$ sudo cp bin/systemd/alignak-webui.service /etc/systemd/system/alignak-webui.service

# Start Alignak WebUI
$ sudo systemctl start alignak-webui

# Reload Alignak WebUI
$ sudo systemctl reload alignak-webui

# Stop Alignak WebUI
$ sudo systemctl stop alignak-webui

# Enable Alignak WebUI to start on system boot
$ sudo systemctl enable alignak-webui
```

Warning: The default logger configuration will make log available on the console. In service mode the console log will be pushed to the */var/log/messages* file and it is quite verbose :)

2.4.4 Environment variables

If an environment variable *ALIGNAK_WEBUI_CONFIGURATION_FILE* exist, the file name defined in this variable takes precedence over the default files list.

If an environment variable *ALIGNAK_WEBUI_UWSGI_FILE* exist, the *alignak-backend-uwsgi* script will use the file name defined in this variable as the uWSGI configuration file.

2.4.5 Developer mode

To run in developper mode (single threaded Web server with few connections), you can start the application with:

```
alignak-webui
```

The default configuration parameter makes the application start on your localhost, port 5001, so you can point your Web browser to:

```
http://localhost:5001/
```

To gain more control on the application start:

```
cd alignak_webui
./app.py -n 0.0.0.0 -b http://127.0.0.1:5000 -d ../etc/settings.cfg
```

All the command line options:

```
./app.py -h
```

2.5 Using the application

2.5.1 Login / logout

Assuming the default configuration has not changed, you can point your Web browser to:

```
http://localhost:5001/
```

The application login page requires that a username and a password are typed. The user authentication is made by the Alignak backend with user accounts which have been previously created in the backend.

With the backend default configuration, it should exist an *admin* user whose password is *admin*.

2.5.2 User session

Assuming the default configuration has not changed, the user session will be alive as long as the logged-in user's web browser is not closed.

2.5.3 Livestate

The livestate page displays the current system live state

2.5.4 Dashboard

To be completed ...

2.5.5 Elements

To be completed ...

2.5.6 Tactical views

To be completed ...

2.5.7 Detailed views

To be completed ...

2.6 Specific parameters

2.6.1 Hosts custom variables

Host location

The Web UI allow to disply the monitored hosts on a map. To achieve this, each host should have GPS coordinates to get located properly.

Hosts fetched from the Alignak backend have a *location* GeoJson attribute that contains the host GPS coordinates. This position is used to locate the host on a map.

Specific custom variables are used to set the host position:

```
# GPS
   _LOC_LAT              45.054700
   _LOC_LNG              5.080856
```

Note that the WebUI is using the Alignak backend as a data backend. The *LOC_LAT* and *LOC_LNG* custom variables are handled by the `alignak-backend-import` script when it imports the hosts into the backend. From those variables, this script creates a GeoJson Point position.

Fix actions

All the host/service custom variables are displayed in the *Configuration* tab of the host/service view.

Some specific custom variables are handled by the Web UI:

- *DETAILLEDESC* contains the detailed description of the host/service
- *IMPACT* contains the description of the impact of an host/service failure
- *FIXACTIONS* contains the description of the actions that can be launched to fix a problem

Those variables, if they exist, are displayed in the host/service overview panel.

Notes and URLs

In an host/service configuration, it is possible to define *notes*, *notes_url* and *action_url*. This is how the Web UI uses these properties.

Each url may be formatted as:

- url,,description
- title::description,,url
- title,,icon::description,,url

description is optional

If *title* is not specified, a default title is used as title If *icon* is not specified, a default icon is used as icon

Some examples:

```
notes                 simple note
notes                 Label::note with a label
notes                 KB1023,,tag::<strong>Lorem ipsum dolor sit amet</strong>,␣
↪consectetur adipiscing elit. Proin et leo gravida, lobortis nunc nec, imperdiet␣
↪odio. Vivamus quam velit, scelerisque nec egestas et, semper ut massa. Vestibulum␣
↪id tincidunt lacus. Ut in arcu at ex egestas vestibulum eu non sapien. Nulla␣
↪facilisi. Aliquam non blandit tellus, non luctus tortor. Mauris tortor libero,␣
↪sagittis quis ultrices in, sollicitudin et tortor.|note simple|Tag::tagged note ...
```

(continues on next page)

(continued from previous page)

```
notes_url             http://www.my-KB.fr?host=$HOSTADDRESS$|http://www.my-KB.fr?host=
↪$HOSTNAME$

action_url            http://www.google.fr|url1::http://www.google.fr|My KB,,
↪tag::http://www.my-KB.fr?host=$HOSTNAME$|Last URL,,tag::<strong>Lorem ipsum dolor␣
↪sit amet</strong>, consectetur adipiscing elit. Proin et leo gravida, lobortis nunc␣
↪nec, imperdiet odio. Vivamus quam velit, scelerisque nec egestas et, semper ut␣
↪massa.,,http://www.my-KB.fr?host=$HOSTADDRESS$
```

2.7 API

2.7.1 Internal routes

The application, as a Web application, manages routes. Some of them are application internal routes and most of them are provided by the application plugins.

The application provided routes:

- /, to get the home page
- /ping, used by the page refresh process
- /login (GET), for the login form
- /login (POST), for the login form
- /logout, to log out the current user
- /static/, to get the static pages (eg. js, css, ...)
- /modal/, to display the modal dialog box with a specific content

2.7.2 Plugins routes

Each plugin defines routes that are added to the application routes when the plugin is loaded.

Most plugins are dedicated to a specific backend element (eg. host, service, ...). For those plugins, some rules, implemented in the Plugin class, are commonly used for the routes:

- */elements*, get the paginated elements list
- */elements/tree*, get the element tree view (for some elements)
- */elements/table*, get the element table view
- */elements/table_data*, get the element table data (called by the datatable)
- */elements/templates/table*, get the element templates table view
- */elements/templates/table_data*, get the element templates table data (called by the datatable)
- */elements/list*, get all the elements list as a json list of objects containing *id*, *name* and *alias*
- */elements/templates/list*, get all the elements templates as a json list of objects containing *id*, *name* and *alias*
- */element/id*, get the view of a specific element. *id* may be the element *id* or *name*

Where *element* stands for the specific element name: host, service, user, ...

Get elements page

Most of the elements plugins provide a paginated view of the elements. Those views are not often used in the Web UI ... except when they are included in some more complex views (eg. host view for the services)

> /services, will display the list of services of an host /hosts, will display the list of hosts /hosts/templates, will display the list of hosts templates and will allow to create a new host

Reload element configuration

Most of the plugins have a configuration file used to define their table structure. Reloading the configuration file dedicated to an element is as easy as:

```
/element/settings
```

This endpoint reloads the plugin configuration file and it alos displays this configuration in a JSON format.

Get element page

Some plugin provide a view for an element. This view / page is available on this endpoint:

```
/host/_id

/host/name
```

Get elements table

This is the most common route provided by each plugin. It allows to display a table view for some elements.

Table view:

```
/hosts/table

/hosts/table?search=

- `?search=` to clear all the table filters
- `?search=name:value` to search for `value` in the column `name`
- `?search=name:value name2:value2` to search for `value` in the column `name` and␣
↪`value2` in `name2`
```

Get elements templates table

This route is provided by plugins associated to templated elements (eg. host, service or user). It allows to display a table view for the element templates.

Templates table view:

```
/hosts/templates/table

/hosts/templates/table?search=

- `?search=` to clear all the table filters
- `?search=name:value` to search for `value` in the column `name`
- `?search=name:value name2:value2` to search for `value` in the column `name` and␣
↪`value2` in `name2`
```

Get elements tree

If the elements are linked together (eg. groups) a tree route will display a tree view of the elements.

Tree view:

```
/hostgroups/tree
```

Get elements list

A JSON list of the elements and elements templates is available on the endpoint */elements/list* and */elements/templates*. If a templates URL parameter (GET or POST) exists, the elements list is completed with the templates list to get all the elements and templates.

List view:

```
/hostgroups/list

/hostgroups/templates/list

/hostgroups/list?templates=1
```

2.7.3 External access

An external application can embed some Alignak WebUI widgets and pages.

Authentication

Embedding a part of Alignak WebUI requires an authentication. Provide credentials as a Basic HTTP authentication in the page request. The HTTP request must have an 'Authorization' header containing the authentication. The Alignak WebUI will use this authentication parameters to check authentication on its Alignak backend.

API

URL syntax:

```
GET <alignak_webui>/external/<type>/<identifier>/<action>

where:
    <alignak_webui> is the base url of your Alignak WebUI (eg. http://127.0.0.1:8868)
    <type> = `widget` for a widget, <identifier> is the identifier of the widget
    <type> = `table` for a table, <identifier> is the identifier of the table
    <type> = `list` for a list, <identifier> is optional
    <type> = `host` for an host widget, <identifier> is the identifier of the host

    <action> is an optional required action (used internally for the tables)

    <action> is the host widget identifier if <type> = `host`

URL parameters (GET or POST):
    **page** provides a full HTML page including necessary Css and Js. Suitable for␣
↪embedding the widget or table in an iframe (see hereunder, Embedding mode)

    **links** provides an URL prefix to be used for the WebUI links. As of it, the␣
↪links will be prefixed with this value to allow a *redirected* navigation rather␣
↪than the internal one.
```

The application response content type is always displayable HTML (eg. *text/html*). Even when an error message is provided. As of it the content can always be included in an HTML page or an iframe HTML element. The HTTP status code is :

- 401 (Unauthorized) for an unauthorized access
- 409 (Conflict) for an API error
- 200 (Success) if content is delivered

The application server implements the CORS an, as of it, filters the external access. The Access Control Allow Origin can be configured in the application configuration file thanks to the **cors_acao** parameter.

Lists

The lists returned when using the type *list* are provided as Json.

Embedding mode

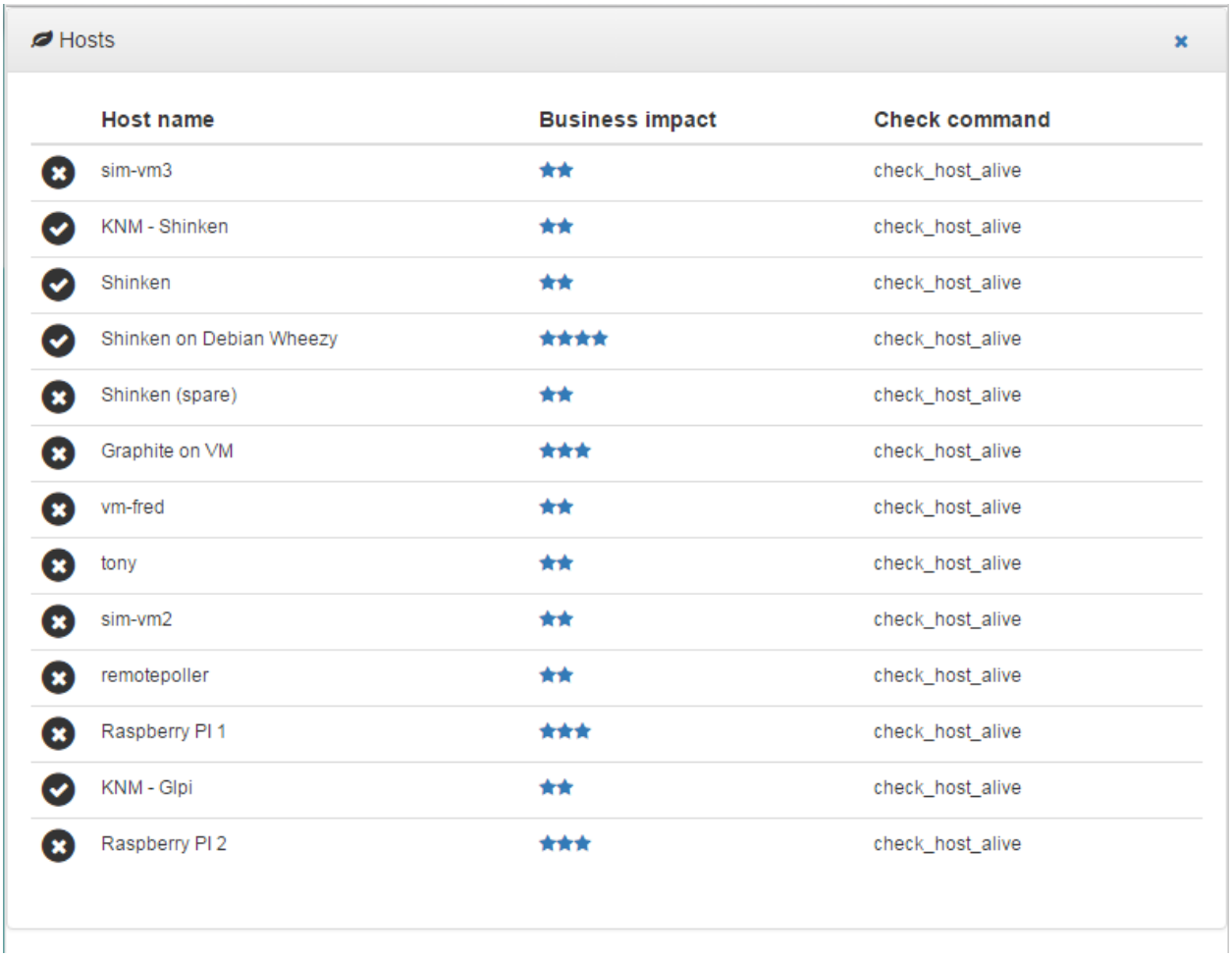
As default, the widget is provided as it is defined in the Alignak WebUI. The widget is an HTML with its content ...

Hosts

Host name	Business impact	Check command
sim-vm3		check_host_alive
KNM - Shinken		check_host_alive
Shinken		check_host_alive
Shinken on Debian Wheezy		check_host_alive
Shinken (spare)		check_host_alive
Graphite on VM		check_host_alive
vm-fred		check_host_alive
tony		check_host_alive
sim-vm2		check_host_alive
remotepoller		check_host_alive
Raspberry PI 1		check_host_alive
KNM - Glpi		check_host_alive
Raspberry PI 2		check_host_alive

Use the URL parameter **page** to get a full page embeddable in an iframe. Without this parameter only the required widget is provided as a text/html response. As an example:

```
http://127.0.0.1:5001/external/host/cogny/view?page
http://127.0.0.1:5001/external/host/cogny/information?page
http://127.0.0.1:5001/external/host/cogny/location?page
```



Please note that in the default mode (no **page** parameter), it is the caller's responsibility to include the necessary Javascript and CSS files. Currently, those files are (at minimum):

```
<link rel="stylesheet" href="/static/css/bootstrap.min.css" >
<link rel="stylesheet" href="/static/css/bootstrap-theme.min.css" >
<link rel="stylesheet" href="/static/css/font-awesome.min.css" >
<link rel="stylesheet" href="/static/css/alignak_webui-items.css" >

<script type="text/javascript" src="/static/js/jquery-1.12.0.min.js"></script>
<script type="text/javascript" src="/static/js/bootstrap.min.js"></script>
```

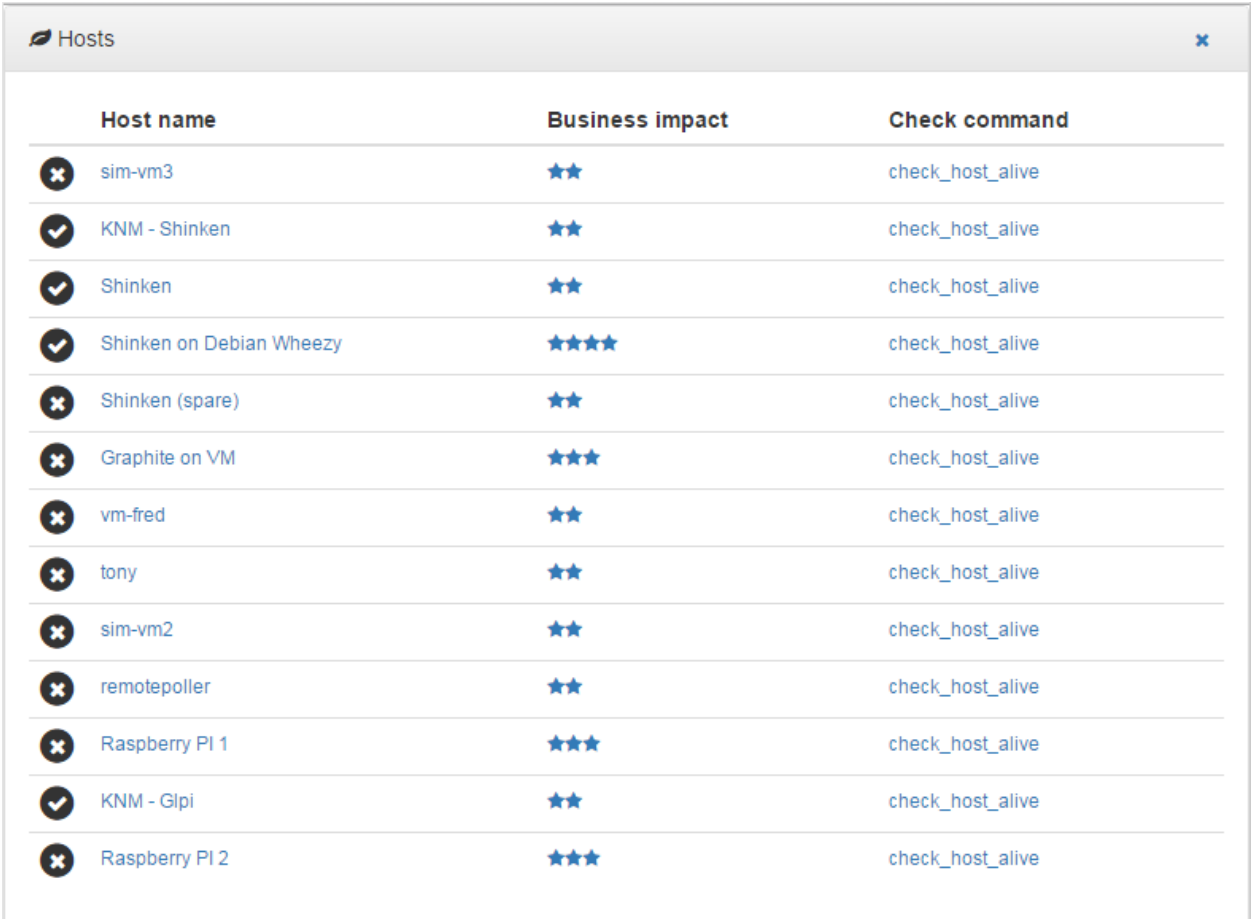
This list is to be confirmed but it should be the right one ;) All the Css and Javascript files (except for Alignak WebUI...) are easily found on major CDNs.

For some external widgets, it is necessary to include also:

```
<!-- Datatables jQuery plugin -->
<link rel="stylesheet" href="/static/css/datatables.min.css" >
<script type="text/javascript" src="/static/js/datatables.min.js"></script>
```

Embedding options

Use the URL parameter **links** to have the navigable links in the embedded page. Else, the links are replaced with their text counterpart.



The **links** parameter must contain the prefix URL used to navigate to the right page for the corresponding link. All links in the widgets are relative from the Web UI home page...

2.8 Developer's documentation

Contents:

2.8.1 Development

Application

Web application developed with Python Bottle micro-framework. See *app.py* for the main application file and *application.py* as the Bottle application and the main routes definition.

User authentication

The application install a *before_request* hook to detect if a session currently exists and an authenticated user is already connected (stored in the session).

If no session exists all the requests are redirected to the */login* page. User is authenticated near Alignak backend with username / password. Once authenticated, a User object representing the current user is stored in the session. This user has an *authenticated* attribute set.

Session management

The application uses Beaker middleware for session management. The configuration is made in *app.py*.

The session is stored in a file to persist across application restarts and to allow using a multi-threaded web server.

A cookie named as the application (Alignak-WebUI) is existing as soon as a session is created. Its default expiry delay is to never expire until the browser get closed.

The session stores the current user and some small other information (login message, application message).

The session behavior may be configured in the application configuration file (see Application configuration file).

Data manager

The application uses a DataManager object to store all the information about the data got from the Alignak backend.

The DataManager is an interface between the application and its plugins, and the data store in the Alignak backend.

TO BE DETAILED !

Datatables

Table configuration

Backend elements can be displayed as a table. For this, the plugin must declare:

- a table build URL (eg. */elements/table*)
- a table update URL (eg. */elements/table_data*)
- a table schema in its configuration file

Note: the table URLs are formed with element endpoint (eg. host), a plural form (add an s) and */table* for the build URL or */table_data* for the update URL.

The plugin class makes this configuration easy and it is enough to define the table configuration.

The *schema* defines the global table configuration and the table columns configuration. The schema is declared in the plugin configuration file. Each column is declared as a section of the configuration file. The declaration order in the configuration file will be used for the column ordering in the table.

The name of each column item much match exactly the name of the backend element field.

The main *table* section in the *schema* is used to configure the global table behaviour. This field allows to define the table title and the table main characteristics (ordering, sorting, ...). See the comments in the example below.

As an example:

```
[table]
; Table global configuration
page_title=Hosts table (%%d items)
visible=True
orderable=True
editable=True
selectable=True
searchable=True
responsive=False

[table.name]
type=string
title=Host name
visible=True
hidden=False
searchable=True
regex=True
orderable=True
hint=This field is the host name
editable=True
required=True
empty=False
unique=True

[table._realm]
title=Realm
type=objectid
searchable=True
allowed=inner://realms/list
resource=realm
visible=False
```

Table parameters

Each table may be:

- visible, (default: True)
- printable, (default: True)
- orderable, (default: True)
- selectable, (default: True)
- searchable, (default: True)
- editable, (default: True)
- responsive, (default: True)

- recursive, (default: False)
- commands, (default: False) - only applies to the livestate table
- css, (default: display)

Initial (default) table sort is defined as (CURRENTLY NOT IMPLEMENTED !):

- initial_sort which is an array of array: [[1, "desc"]]

All the tables are sorted by default on the first defined column by ascending value.

Table css classes are defined here: https://datatables.net/manual/styling/classes

Table display

If a status_property is defined for the table (default is to use the *status* field in the elements), then each table row has an extra CSS class named as: table-row-status_property.

As an example, for the livestate table, an element with status UP will have a CSS class **table-row-up**.

The corresponding classes can be defined in the *alignak_webui-items.css* file. Some example classes still exist in this file for the livestate states (eg. UP, OK, ...).

I a table column has a *visible* attribute defined as False, this column will not be displayed in the table. To hide a column and allow the user to show this column thanks to the table column selector, you can use the *hidden* attribute and set it to True.

Field attributes:

- *visible* (True): to include a column with this field
- *hidden* (False): to hide the column in the table display
- *type*: is the field type (see the known types list hereunder)
- *content_type*: is the list items content type (eg. same as type) if the field is a *list*
- *hint*: is a descrption of the field used as an help in the edition form
- *required*: to indicate if the field must contain a value or may be empty
- *allowed*: the list of the allowed values in the field (see hereafter for more explanations)

Field types:

- *string* (default)
- *integer*
- *float*
- *boolean*
- *objectid*
- *list*
- *dict*
- *point*

Field content types (for a list of items):

- *string* (default)
- *integer*

- *float*
- *boolean*
- *objectid*
- *dict*
- *point*

Available formats:

- *date*:
- *on_off*:
- *single*: only one value is allowed in the list field
- *multiple*: several values are allowed in the list field

When the field *type* is a list, the *content_type* field must specify which type is to be used for the list items (eg. string, integer, ...). If the *allowed* field contains a value, it may be:

- inner://url
- a comma separated list of the allowed values

If the edited item is a template, the *allowed_template* (if it is defined) is used instead of the *allowed* value. This to allow defining a different list of allowed values for the templates.

Table filtering

Table filtering is available on a column basis; each column can have its own search parameter in the table header. The filtering field is an input field, a select field, ... according to the column type/format.

As much as possible, the table column format is determined by the application thanks to the columne *type* parameter.

The column format is used to choose the filtering input method. In some cases, it may be useful to specify the format.

The following rules apply:

- as a default, *format* is **string** which means that the filtering input method is an input field
- when *type* is **list**, the format method will automatically be a *select*. The *allowed* parameter defines the content of the allowed values in the select options.
- *format* can be specified as a *select* (unique value) or *multiselect* (multiple select) input method
- when *type* is **objectid**, the format method will automatically be a *select* that wil be populated with the related object names list

Available formats:

- *date*:
- *on_off*:
- *select*:
- *multiselect*:

The data backend search is made with an AND operator on all the provided values. Furthermore, each column has a *regex* parameter. This parameter indicates wether the search is an exact (False) or loose (True) match on the data value.

The table filtering is stored in the user's preferences to be restored the next time the page is refreshed or browsed.

A table button indicates if some filters are activated and also allows to clear the currently applied filters.

Web UI pages displaying a datatable can receive an URL parameter to influence the data filtering. If the *search* query parameter is present in the URL it takes precedence over the existing column filtering. As of it, the user can request a specific table filter that will be used instead of the saved filtering.

On table loading, the filtering logic is as follows:

- restore previously saved state
- if no URL filtering is present, restore filters from saved state
- if URL filtering is present, clear table filtering and apply URL filtering

The URL filtering parameter *search* has a very simple syntax:

- *?search=* to clear all the table filters
- *?search=name:value* to search for *value* in the column *name*
- *?search=name:value name2:value2* to search for *value* in the column *name* and *value2* in *name2*

Some examples:

- livestate hosts UP: *search=type:host state:UP*
- livestate hosts DOWN: *search=type:host state:DOWN*
- livestate services WARNING: *search=type:service state:WARNING* or *search=type:service state_id:1*
- livestate hosts/services OK/UP: *search=state_id:0*
- livestate elements business impact high: *search=business_impact:5*

HTML templates

TO BE EXPLAINED !

Debug mode

Many templates declare a local *debug* variable that will display extra information. Simply declare this variable as True (eg. *%setdefault('debug', True)*). Debug information panels have a *bug* icon ;)

Some specific templates for debug mode:

- layout.tpl, will display all the HTTP request information
- _actionbar.tpl will display all the widgets available for dashboard and external access

Good practices

From Python to javascript, main javascript variables are declared in layout.tpl to be available for every HTML and Javascript files.

Application UI design

The application User Interface design is based upon Google Material Design served by the Bootstrap Material Design project (https://github.com/FezVrasta/bootstrap-material-design). This project has been forked in the Alignak monitoring contrib organization on our github to make some modifications for the Web UI layout.

The default CSS can be changed and rebuilt from the project LESS files.

On a Linux Ubuntu:

```
sudo apt-get install nodejs-legacy
sudo apt-get install npm
sudo apt-get install node-less

sudo npm install bower -g
sudo npm install -g grunt-cli

git clone https://github.com/Alignak-monitoring-contrib/bootstrap-material-design
cd bootstrap-material-design/
npm install && bower install

# Colors are defined in less/_colors.less
# Change variables in less/_variables.less

grunt less      # Rebuild CSS files in dist/css
grunt cssmin    # Minify CSS files in dist/css
```

Once the new CSS files are built copy the content of the dist/css directory into the htdocs/css/material directory of the WebUI.

2.8.2 Plugins

The application loads plugins... TO BE COMPLETED!

Plugin structure

A plugin is a sub-directory in the application *plugins* directory (eg. *plugin*) which must contain a Python file named *plugin.py* that is imported by the application as a Python module.

All directories in the application *plugins* directory that do not match this requirement are ignored by the application.

The *plugin.py* file must declare a global class inherited from the application Plugin class:

```
class PluginHosts(Plugin):
    """ Hosts plugin """

    def __init__(self, app, cfg_filenames=None):
        """
        Hosts plugin

        Overload the default get route to declare filters.
        """
        self.name = 'Hosts'
        self.backend_endpoint = 'host'

        self.pages = {
        ...
        }

        super(PluginHosts, self).__init__(app, cfg_filenames)
```

If the plugin is dedicated to a specific Alignak backend type of elements, it must declare which element is managed in a property *backend_endpoint*.:

```
# Declare backend element endpoint
self.backend_endpoint = 'host'
```

This will allow the application to route all the request for this type of element to this specific plugin. When an external application requests an hosts list, this plugin will be requested for the list.

If the plugin is to log some information, it must use this pattern:

```
from logging import getLogger
logger = getLogger(__name__)
```

As of it, the plugin logs will be included in the application main log stream.

Plugin configuration file

A plugin can have its own configuration file.

The application searches in several location for a configuration file:

- /usr/local/etc/alignak-webui/plugin_NAME.cfg
- /etc/alignak-webui/plugin_NAME.cfg
- ~/alignak-webui/plugin_NAME.cfg

- ./DIR/settings.cfg

Where NAME is the plugin name and DIR is the plugin directory.

The configuration file is built like an Ini file parsed thank to Python ConfigPaser:

```
; ------------------------------------------------------------------------------
↪------
; Plugin configuration file formatted as RFC822 standard
; ------------------------------------------------------------------------------
↪------

[timeperiods]
; Plugin global configuration
;enabled=False

; A parameter in a section named like the plugin is seen as a direct parameter:
; timeperiods.variable is available as: self.plugin_parameters['variable']

[test]
variable2=2
; A parameter in another section is seen as a dict parameter:
; test.variable2 is available as: self.plugin_parameters['test']['variable2']

; The table and table. sections are specific:
; test.variable2 is available as: self.plugin_parameters['test']['variable2']
[table]
; Table global configuration
page_title=Timeperiods table (%d items)
visible=True
orderable=True
editable=True
selectable=True
searchable=True
responsive=False
recursive=True

[table.name]
title=Timeperiod name
type=string
searchable=True
regex=True
orderable=True
editable=True
hint=This field is the time period name
```

Once parsed, the configuration file will make available an ordered dictionary in the plugin class: `self.plugin_parameters`. The `self.plugin_parameters['table']`, also aliased as `self.table`, contains the table structure. Using the *element/settings* route with a Web browser will output Json formatted data with the parameters.

Plugin table configuration

A plugin can have its own configuration file.

Whole table configuration

```
; Table global configuration
[table]

; Items page title - used when displaying items table
page_title=Hosts table (%d items)

; Templates page title - used when displaying templates table
template_page_title=Hosts templates table (%d items)

; Obviously ;)
visible=True

; The table may be printed
printable=True

; The table may be ordered - then orderable fields are active
orderable=True

; The table is editable - items can be selected for edition
editable=True

; The table is selectable - rows can be selected
selectable=True

; The table is searchable - searchable fields are active
searchable=True

; The table is responsive or not - responsivenes adds an horizontal bar
responsive=False

; The table is recursive (sic)- can navigate to a tree view
recursive=True
```

Table field configuration

```
; Declare the field 'name' of the table
[table.name]
; Title of the table column
title=Timeperiod name
type=string

; When displaying the templates table, only the fields having templates_table=true␣
↪are displayed
templates_table=true

; This field is searchable
searchable=True
; If regex is true, search in the table with a regex, else search for strictly␣
↪identical content
regex=True

; The table may be ordered
orderable=True
```

(continues on next page)

(continued from previous page)

```
; Edition part
; ------------
; This field is editable
editable=True
; The hint information is displayed in the edition form to explain the field content
hint=This field is the time period name
; Required field
required=true
; Field can be left empty or not
empty=false
; Must contain a unique value
unique=true
```

Plugin routes

A plugin may declare routes for the application Web server. The routes declaration is made through a global dictionary named *pages*.

Main routes:

- elements view: /elements
- elements table: /elements_table
- elements list: /elements_list
- elements templates: /elements_templates
- element: /element/element_id
- elements widgets: /elements/widget
- element widget: /element/element_id/widget_id

For a recursive element (eg. hostgroups, . . .):

- elements tree view: /elements_tree

A complete example of what is possible can be found in the **hosts** plugin. The source code is commented to explain what is done. . .

2.8.3 Widgets and tables

The Alignal WebUI proposes several widgets and tables.

Widgets

Hosts table

Displays a simple list of the monitored hosts with their current status, business impact and check command.

Options:

- number of elements
- filter on host name and alias

- filter on host parameters

Hosts chart

Displays a pie chart for the monitored hosts with their current status.

Services table

Displays a simple list of the monitored services with their current status, business impact and check command.

Options:

- number of elements
- filter on service name and alias
- filter on service parameters

Services chart

Displays a pie chart for the monitored services with their current status.

Tables

All the elements have a table to display them. The table name is build with this simple rule: name_table. As examples:

- hosts_table
- services_table
- logcheckresults_table
- ...

Depending upon the element table configuration in its plugin, the table is searchable, orderable, ...

Host widgets

The host information page is built with *host widgets* declared in the hosts plugin. Each host widget is included in a tab of the host page navigation tab control.

Available host widgets are:

- information
- configuration
- services
- timeline
- history
- metrics
- location

2.8.4 Application interface layout

Material design:

- *static/css/material* directory contains the files used to configure the material look and feel of the application. Those files may be changed with the result of the rebuild explained in the develop part of this documentation (see Application UI design).

Css files:

- *alignak_webui.css*, contains the main classes used by the Web UI
- *alignak_webui-items.css*, contains the CSS classes used for the items icons styles as declared in the application configuration file (see hereunder)

Javascript files:

- *alignak_webui-layout.js*, contains some colors definitions for the externally embedded widgets