Learner Profiler: ASR's Final Major Project

This document contains information on how to install and start this project.

The project is running under Symfony Standard Edition - a fully-fonctional Symfony2 application.

Installation

1) Downloading Dependencies

As Symfony uses Composer to manage its dependencies, the recommended way to create a new project is to use it.

If you don't have Composer yet, download it following the instructions on http://getcomposer.org/ or just run the following command:

```
curl -s http://getcomposer.org/installer | php
```

Then, use the `update` command to download all dependencies.

2) Define VirtualHost (Optional)

If you are using apache2 you can use a virtualhost by create a configuration file *ex: `/etc/apache2/sites-available/leaner_profiler.conf`*

```
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    ServerName learnerprofiler.dev
    ServerAlias www.learnerprofiler.dev

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/learnerprofiler/
    <Directory />
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    LogLevel debug

    ErrorLog ${APACHE_LOG_DIR}/learnerprofiler_error.log
    CustomLog ${APACHE_LOG_DIR}/learnerprofiler_access.log combined

    # For most configuration files from conf-available/, which are
```

```
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
</VirtualHost>
```

Then you have to enable this virtual host by using the command `a2ensite leaner_profiler.conf`.

Finally restart your server by using command `service apache2 restart`.

3) Checking your System Configuration

Before starting coding, make sure that your local system is properly configured for Learner Profiler and especially for Symfony2.

Execute the `check.php` script from the command line:

```
php app/check.php
```

The script returns a status code of `0` if all mandatory requirements are met, `1` otherwise.

Access the `config.php` script from a browser:

```
http://learnerprofiler.dev/web/config.php
```

If you get any warnings or recommendations, fix them before moving on.

4) Configuration and Setup

Setting up Permissions

One common issue is that the app/cache and app/logs directories must be writable both by the web server and the command line user. On a UNIX system, if your web server user is different from your command line user, you can run the following commands just once in your project to ensure that permissions will be setup properly.

1. Using ACL on a system that supports chmod +a

Many systems allow you to use the chmod +a command. Try this first, and if you get an error - try the next method. This uses a command to try to determine your web server user and set it as HTTPDUSER:

```
$ HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-data|[n]ginx' | grep -v root | head -1
| cut -d\  -f1`
$ sudo chmod +a "$HTTPDUSER allow delete,write,append,file_inherit,directory_inherit" app/cache
app/logs
$ sudo chmod +a "`whoami` allow delete,write,append,file_inherit,directory_inherit" app/cache
app/logs
```

2. Using ACL on a system that does not support chmod +a

Some systems don't support chmod +a, but do support another utility called setfacl. You may need to enable ACL support on your partition and install setfacl before using it (as is the case with Ubuntu). This uses a command to try to determine your web server user and set it as HTTPDUSER:

```
$ HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-data|[n]ginx' | grep -v root | head -1
| cut -d\  -f1`
$ sudo setfacl -R -m u:"$HTTPDUSER":rwX -m u:`whoami`:rwX app/cache app/logs
$ sudo setfacl -dR -m u:"$HTTPDUSER":rwX -m u:`whoami`:rwX app/cache app/logs
```

If this doesn't work, try adding -n option.

3. Without using ACL

If you don't have access to changing the ACL of the directories, you will need to change the umask so that the cache and log directories will be group-writable or world-writable (depending if the web server user and the command line user are in the same group or not). To achieve this, put the following line at the beginning of the app/console, web/app.php and web/app_dev.php files:

```
umask(0002); // This will let the permissions be 0775

// or

umask(0000); // This will let the permissions be 0777
```

Note that using the ACL is recommended when you have access to them on your server because changing the umask is not thread-safe.

4. Use the same user for the CLI and the web server

In development environments, it is a common practice to use the same unix user for the CLI and the web server because it avoids any of these permissions issues when setting up new projects. This can be done by editing your web server configuration (e.g. commonly httpd.conf or apache2.conf for Apache) and setting its user to be the same as your CLI user (e.g. for Apache, update the User and Group values).

Setting up parameters

You have to copy the `app/config/parameters.yml.dist` to `app/config/parameters.yml` and edit parameters :

```
# This file is a "template" of what your parameters.yml file should look like
parameters:
    database_driver:   pdo_mysql
    database_host:     127.0.0.1
    database_port:     ~
    database_name:     symfony
    database_user:     root
    database_password: ~

    mailer_transport:  smtp
    mailer_host:       127.0.0.1
    mailer_user:       ~
    mailer_password:   ~

    locale:            en

    # A secret key that's used to generate certain security-related tokens
    secret:            ThisTokenIsNotSoSecretChangeIt

    debug_toolbar:          true
    debug_redirects:        false
    use_assetic_controller: true
```

```
#Your ecampus login
ecampus_username: ~ #Your Ecampus username
ecampus_password: ~  #Your Ecampus password
```

Enjoy !

What's inside?

Symfony Standard Edition

The Symfony Standard Edition is configured with the following defaults:

- Twig is the only configured template engine;
- Doctrine ORM/DBAL is configured;
- Swiftmailer is configured;
- Annotations for everything are enabled.

Symfony Standard Edition comes pre-configured with the following bundles:

- **FrameworkBundle** - The core Symfony framework bundle
- **SensioFrameworkExtraBundle** - Adds several enhancements, including template and routing annotation capability
- **DoctrineBundle** - Adds support for the Doctrine ORM
- **TwigBundle** - Adds support for the Twig templating engine
- **SecurityBundle** - Adds security by integrating Symfony's security component
- **SwiftmailerBundle** - Adds support for Swiftmailer, a library for sending emails
- **MonologBundle** - Adds support for Monolog, a logging library
- **AsseticBundle** - Adds support for Assetic, an asset processing library
- **WebProfilerBundle** (in dev/test env) - Adds profiling functionality and the web debug toolbar
- **SensioDistributionBundle** (in dev/test env) - Adds functionality for configuring and working with Symfony distributions
- **SensioGeneratorBundle** (in dev/test env) - Adds code generation capabilities

All libraries and bundles included in the Symfony Standard Edition are released under the MIT or BSD license.

Licence

TODO: Licence file