GNU social - V3 rewrite with Symfony

Hugo Sales

```
Name: Hugo Sales
Email: hugo@fc.up.pt
Project Name: GNU social
Timezone: GMT
Other contact: someonewithpc@freenode.net#social
Proof of Competence: notabug.org/someonewithpc/gnu-social
```

Summary

GNU social is a communication software used in federated social networks, however, it relies on outdated and unmaintained libraries. To this end, this project will rewrite the core code in the Symfony framework.

Benefits

This project will serve as a foundation for a complete reworking of the core code. This is necessary because social has been falling behind other federated software with it's somewhat outdated code and less than ideal performance. However, the general architecture as well as it's philosophy, event system, being extensible, customizable, accessible, Any Browser compliance, the ability to run in a shared hosting environment, among other qualities, makes social a worthwhile project to update, and revitalize. Further, there is still a sizable community of users, who are very excited with the recent changes.

Deliverables

This project will:

- Remove all unmaintained dependencies
- Rebuild the application atop a modern framework
- Port the ORM model to Doctrine
- Simplify the way modules and plugins are handled
- Add a permission system for plugins, such that they can be prevented from using certain events, or accessing certain tables.
- Extensive code coverage and testing

Relevant research

A huge problem that was noted by previous GNU social contributors was the current ORM. According to Diogo, it's something he and XRevan86 struggled with last summer. There was a relative consensus in IRC that we really have to replace it with something like Doctrine. Serious efforts were done to maintain both PEAR and the current

ORM. Ultimately, it was concluded that it wasn't worth it. Changing the ORM to some extent implies, and certainly becomes easier with a change of framework, allowing us to offload some of the maintaining.

There are some tests, but they're few and far between, barely covering only a few of social's components. A big goal of this project is to do test-driven-development from the start, making further maintaining easier.

There are, however, some really good parts to social's architecture, among which are it's event system, which allows for a large degree of decoupling between components, having a more defined interface; another good part is it's extensibility through plugins, facilitated in large part by the aforementioned event system. This is be preserved and built upon.

Database

Currently, the ORM model is a custom implementation based on the outdated and unmaintained PEAR DB library, which required extensive editing last summer, in order to reintroduce support for PostgreSQL.

The downside of the current approach, is that the whole database definition and plugins are checked on each request. Symfony provides a "compiler", which allows doing tasks like these only once.

Cache

Currently, the user is encouraged to enable PHP's OPCache, which caches the bytecode. At the application level, there are plugins for Memcached, Redis, a custom disk based, and a in-process-memory solution.

Support for Memcached will be maintained, while support for Redis is expected to be greatly expanded upon.

Queues

There are currently three queuing strategies: - no queue, where processing is done in-place, through the `UnCache` plugin - in-process queue, where processing is done at idle time - queue daemon, where a separate process handles the queue processing

Additionally, for the last two strategies, there are three implementations: a database table holding work units, through the `DBQueue` plugin, a Redis based and a generic STOMP based solutions.

Core path handlers

Streams

There are a variety of streams, representing each feed in formats like HTML, RSS and ActivityStreams1.0-json. Currently the performance of these is indered by inadequate caching, which will be fixed with the use of Redis.

Admin control panel

The control panel is a semi-finished approach to managing the application through the web browser, mainly for enabling and disabling plugins.

In V3, there will no longer be a `config.php`, as it causes a two generals problem, since the configuration is shared between the aforementioned file and a database table. The latter will be used in V3.

Authentication

Naturally, there is a user authentication system, which allows for in-website, as well as OAuth 1.1a and OpenID mediated login.

In V3, we'll drop support for OAuth 1.1a and transition to OAuth 2, as well as replace the old system with the one provided by Symfony.

Modules

To achieve a greater isolation between components, core functionality is separated among modules, which communicate through the Event system. This is a great solution and will be maintained. The same goes for plugins, however, these are optional.

This system is already ported over in my proof of competence, however there is still a lot of work left porting each individual module and plugin.

Why Symfony

The current codebase is centered on PEAR, which is a an outdated set of libraries, so it was clear it had to change. The choice, ultimately, came down to mainly either Symfony or Laravel.

Both are capable and mature frameworks, but given Symfony's longer track record, it seemed initially like a more appropriate choice. After reading about and examining both frameworks, I concluded that while Laravel aims to be, in a way, easier to use, that could be really limiting for our use case, since we aim to not couple the existing codebase too tightly to any framework, and instead implement our old utilities on top of it. Additionally, while the licenses are the same, Laravel seemed to somewhat focus on Apple's macOS, providing specialized tools for this platform which seemed to be restricted to it. Additionally, the documentation was sparser.

With all of this in mind, and the agreement of Diogo, I proceeded to use symfony and implement a proof of concept.

Plan

Big Crunch and Big Bang

I started my proof of concept by deleting all the existing files. This allowed me to work on a clean slate, but keep the long history of the project, and import old parts, piece wise.

Proof of competence Walkthrough

The code flow starts with a request to the `index.php` file. This is the entry point for the remote access to the application. There is also a command line utility provided by symfony which differs mostly only on this step, but provides a wide variety of tools to help develop, deploy, and manage the application.

In `index.php` an instance of `src/Kernel.php` is created, which is responsible for initializing and holding all the components of the Symfony framework. In it, we define our constants, like `INSTALDIR`, as well as register a `CompilerPass`, which, as the name suggests, is a piece of code which is run in the compilation phase of the framework. We use this to define a new Doctrine 'metadata driver', i.e., a method of defining our tables. This is necessary because we want to maintain our old method of using a static `schemaDef` method, with a determined syntax, in order to make future possible migrations, and code reusability easier. Next, `src/Util/GNUsocial.php`, which implements the `EventSubscriberInterface` is called, through Symfony's autowiring, on either a `kernel.request` or a `console.command` event. These events occur just after the initialization of the framework, but before the request is handled, allowing us to introduce our main application entry point. This class is responsible for collecting, through Symfony's Dependency Injection, references to all the Symfony services we need to use throughout the application, with the goal of making them available statically, once again allowing us to not drastically change our architecture. It's also here the new `Module Manager` is initialized, a class which is responsible for handling

the loading and registration of modules and plugins through a manifest, linking their event handlers, handling permissions, establishing their routes, controllers and entities. From here, the code flows mostly as it did before, where the router calls the appropriate controller which can use events and everything else to accomplish it's goal.

Tentative timeline

The following is an estimate of the project timeline, subject to changes, which will be communicated with and agreed upon by my mentors.

The event system, i18n/l10n as well as some other small components are already implemented.

June 1st - 8th

Basic test structure Port database definitions Implement configuration

June 9th - 15th

Restructure the database Port installer

June 16th - 22nd

Implement custom symfony response wrapper, such that each controller is only responsible for returning the raw data, and said wrapper formats it in the requested format (HTML, RSS, JSON).

June 23rd - 29th

Port old relevant modules

June 30th - July 3rd – 1st Evaluation

Continue modules port Evaluation

July 4th - 13th

Implement main feed Session system Side panels

July 14th - 20th

Feeds and streams

July 21st - 26th

Write docker configuration script

July 27th - 31st – 2nd Evaluation

Implement queuing system Evaluation

August 1st - 10th

Verify and complete tests

August 17th - 23rd

Buffer week

August 24th - 31st – Final week

Verify documentation, tests, coverage, docker images

Communication

My code will be hosted on a fork of Diogo's fork, hosted in notabug.

Working on a separate repository, will allow me to make and edit commits, without exposing users to major history changes, since a significant part of the community has switched to Diogo's repository.

Further, communication will be made through IRC, on the `#social` channel on freenode, where my nickname is `someonewithpc`.

I will communicate my progress at least twice a week, besides asking for clarifications.

Qualification

GNU social is a project I really believe in. I'm a strong supporter of the GNU philosophy. While other projects seem to only be interested in the newest and shiny things, social plans to stick to supporting OStatus, Any Browser, among other accessibility features which make the software not only more versatile, but more welcoming of new users.

I am rather experienced with Bash, PHP, SQL. Furthermore, I am currently enrolled in my third year of Computer Science studies at the Faculty of Sciences of the University of Porto, Porto (Portugal).