GOTIT 1.1 Web application documentation for administrator: installation and configuration

22/07/2019 -v1.1

Philippe Grison[1] | Florian Malard[2] | Louis Duchemin[2] | Lara Konecny-Dupré[2] | Tristan Lefébure[2] | Nathanaëlle Saclier[2] | David Eme[3] | Chloé Martin[1] | Cécile Callou[1] | Christophe J. Douady[2]

(1) BBEES - Unité Bases de données sur la Biodiversité, Écologie, Environnement et Sociétés (BBEES), Muséum national d'Histoire naturelle, CNRS

(2) LEHNA : UMR CNRS 5023 Ecologie des Hydrosystèmes Naturels et Anthropisés, Université Lyon 1 , ENTPE, CNRS, Université Lyon

(3) New Zealand Inst. for Advanced Studies, School of Natural and Computational Sciences, Massey Univ., Auckland, New Zealand

Table of Contents

I – GOTIT project

I.1 - Introduction

The GOTIT project is a collaboration between software engineers (BBEES) and biodiversity researchers (LEHNA) which aims at developing a database structure (db_gotit1) and related web application (GOTIT) for optimizing the input and management of data, metadata and vouchers produced on a-day-by-day basis by biodiversity laboratories involved in the delimitation, inventory and distribution of species. The tool is particularly suitable for designing and monitoring biodiversity projects that employ an integrative taxonomic approach combining morphology-based and DNA-based species occurrence data to document and explain species distribution patterns.

The present documentation describes the necessary steps for installing GOTIT Web application. The description is for installing GOTIT either on a local or remote server. Detailed information on how to use GOTIT once installed is available at https://github.com/GOTIT-DEV/GOTIT. A demo version of GOTIT is also available at https://gotit.cnrs.fr.

I.2 - The GOTIT Web application

The GOTIT web application was developed with the Symfony framework (https://symfony.com/) and the Doctrine (https://www.doctrine-project.org/) Object Relational Mapper (ORM). Symfony is one of the most popular PHP frameworks for developing large-scale enterprise web applications. It provides a large set of reusable PHP libraries that can assist with the completion of tasks such as templating, authentication, routing, object configuration, and form creation. Symfony makes heavy use of existing PHP open-source projects as part of the framework, including:

- Doctrine as object-relational mapping layers
- PDO database abstraction layer (1.1, with Doctrine)
- PHPUnit, a unit testing framework
- Twig, a templating engine

The framework implements the “Model View Controller” (MVC) architecture pattern which separates out the application logic into three separate parts, promoting modularity and ease of collaboration and reuse.

- The application model describes all the data that the application must contain. All information about the data is managed via the ORM Doctrine which allows to Create, Read, Update and Delete (CRUD) data from the database (db_gotit1).
- The view defines how the application's data should be displayed.

- The controller contains logic that updates the model and/or view in response to input from the users of the application.

The different steps followed and main components called when submitting a GOTIT Web page are described in Figure 1.

- The http request (for example, a web form) is processed by the Symfony front-end controller which calls the Symfony kernel containing routing and security processing.
- The Controller calls first the Doctrine Model manager and then the form component that defines the data to be rendered in the view template.
- At last, the controller gives an HTML response, that is to say, a web page to the browser.

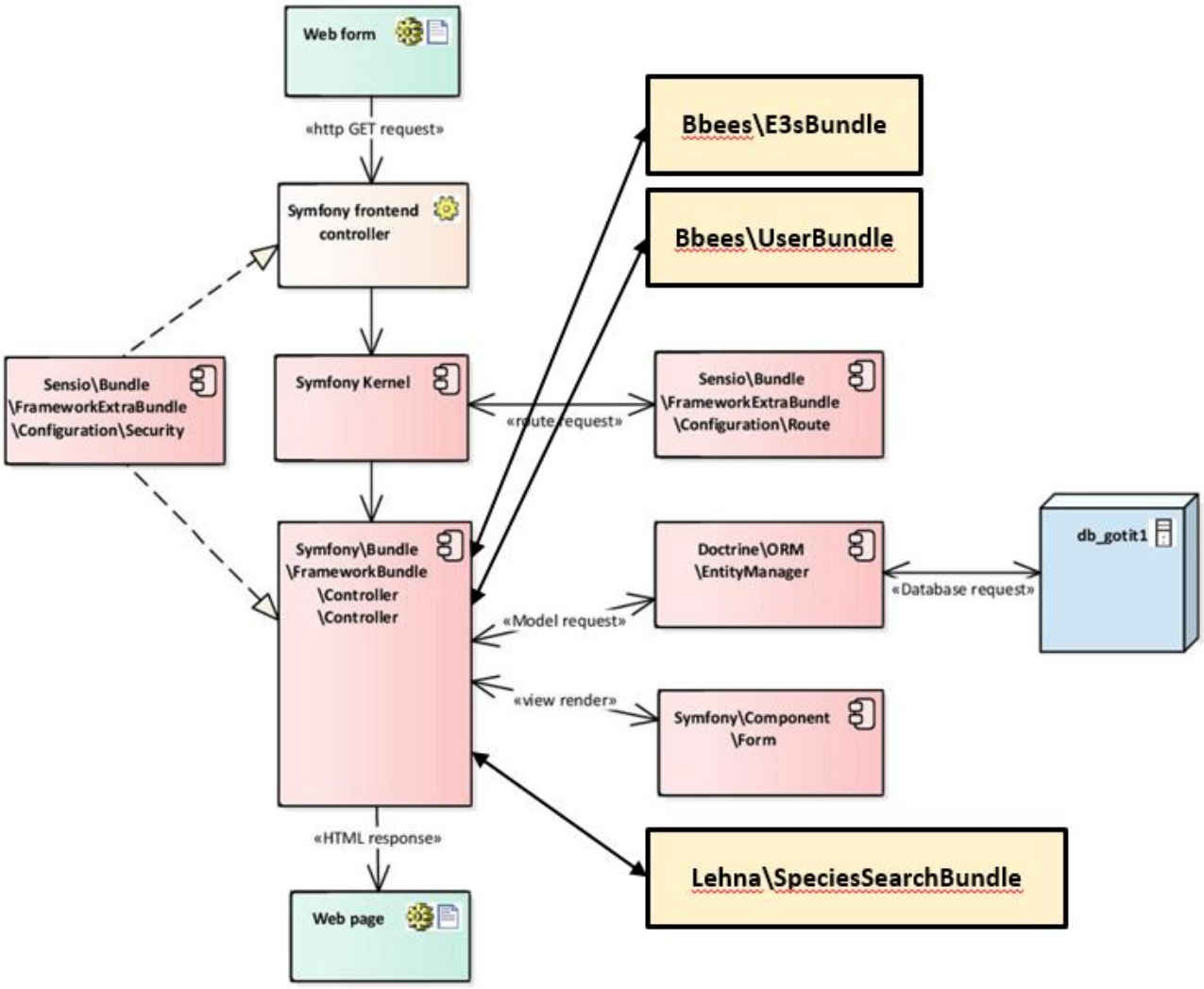

Figure 1: The GOTIT architecture. Red boxes: Symfony / Doctrine components; blue box: GOTIT database; green boxes: web pages; orange boxes: GOTIT bundles. Plain double arrows: Request and Response; plain single arrows: Request or Response; dotted single arrows: embedded components.

I.3 – The database structure

We provide below a brief description and simplified structure of the relational data base in which GOTIT stores and retrieves information (a detailed description of the data base is provided in the file entitled GOTIT_help_database.pdf) (Figure 2). The database structure provides the necessary traceability for recovering the set of methods and biological material linked to any species occurrence. The logical data model has five distinct pathways. The first pathway portrays a species-occurrence data-production process as classically followed by many research laboratories. As part of this process, a site is visited (i.e. box 'Site' in Figure 2) and sampled ('Sampling'), thereby providing specimen lots ('Internal biological material'). Then, specimens are isolated from a lot (Specimen) and used for dissecting and mounting ('Slide') and/or producing DNA sequences ('DNA', 'PCR', 'Chromatogram', 'Internal sequence'). The second pathway allows to trace the storage location of all vouchers produced during laboratory work including specimen lots, specimen slides and DNA extracts. The third pathway accommodates data from the literature, either species occurrence data ('External biological material') or DNA sequence data ('External sequence'), to which we refer as to external data. The fourth pathway allows to link biological materials and sequences (either internal or external) to literature references ('Source'). The fifth pathway consists in assigning morphology-based and DNA-based species hypotheses (SHs) to occurrence data. Morphology-based SHs ('Taxon') are assigned to biological material, specimens and sequences and the criterion used for assignment is specified in the table 'Identified species'. We refer to DNA-based SHs as molecular operational taxonomic units (MOTU). 'MOTU numbers' are assigned to DNA sequences and the sequence data set used to delimit MOTUs are described in the table 'MOTU'.

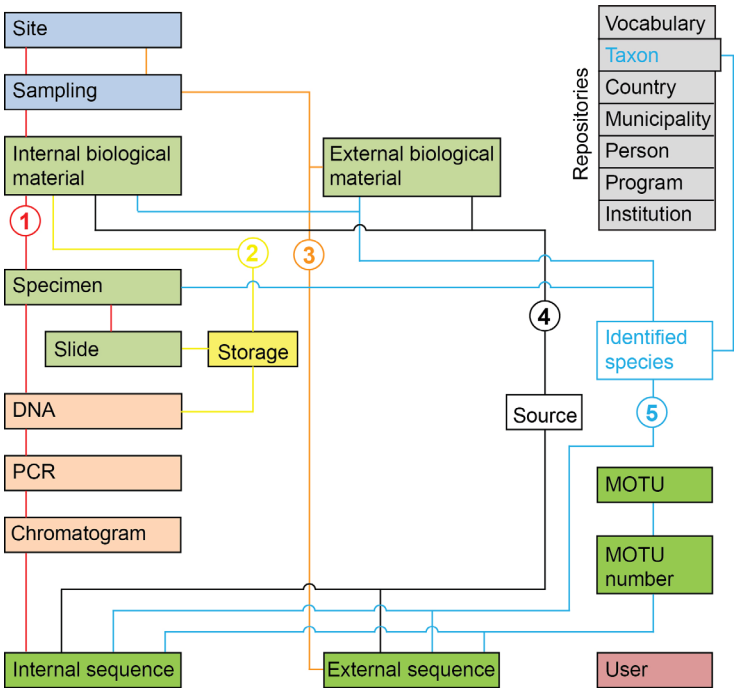

Figure 2. Simplified schematic model of GOTIT database. Numbered and colorized lines correspond to distinct pathways. 1) red: internal data production; 2) yellow: storage; 3) orange: external data; 4) black: data sources; 5) blue: species assignment. Repositories act as dictionaries containing terms used to fill in properties of tables and the user table enables to set up user privileges.

I.4 – Requirements and technologies

Mozilla Firefox (version ≥ 47.0) or Google Chrome (version ≥ 71.0) is required to operate GOTIT web application. GOTIT does not operate with Internet Explorer. The application was designed to be released as a free and open source software (GNU license). We used:

- The database management system Postgresql (version 9.5);
- The Long-Term Support version of Symfony (v3.4 – LTS) with the ORM Doctrine (v2.5);
- The language PHP (v7.1);
- The JavaScript and CSS libraries listed in Table 1.

Table 1. JavaScript and CSS libraries used in GOTIT 1.1

Library	Version	Licence	Usage	Web site
Bootstrap	3.3.6	MIT	front-end framework for developing responsive	getbootstrap.com
JQuery	2.2.4	MIT	feature-rich JavaScript library	jquery.com
jQuery Bootgrid	1.3.1	MIT	dynamic display of HTML tables	www.jquery-bootgrid.com
font-awesome	4.6.3	CC-BY4.0 SIL-OFL1.1 MIT	font and icon toolkit based on CSS and LESS	fontawesome.com
Datatables	*1.10.15*	*MIT*	dynamic display of HTML tables	www.datatables.Net
Mustache	2.3.0	MIT	Templating library	github.com/janl/mustache.js
Moment	2.13.0	MIT	Date-time manipulation	github.com/moment/moment.js
Bootstrap-select	1.12.4	MIT	Enhanced select inputs	silviomoreto.github.io/bootstrap-select
Bootstrap-toggle	2.2.1	MIT	Checkbox styling as switches	www.bootstraptoggle.com
Plotly.js	1.35.2	MIT	D3.js based charts library	plot.ly/javascript/

I.5 – Security

The security of the application is based on the security mechanisms of the Symfony framework (https://symfony.com/doc/current/components/security.html). Parameters related to the settings of the firewall, the roles, and the management of the access form are entered in the security.yml configuration file.

Management rights are based on the following principles:

- Application-level access rights are managed according to four user roles: ROLE_ADMIN, ROLE_PROJECT, ROLE_COLLABORATION and ROLE_INVITED.
- The ROLE_INVITED provides access to the data in the READ mode only. Users have access to the species search tools but they cannot export results.
- Users with the ROLE_COLLABORATION have CRUD rights (create, read, update and delete) on their own data but only a read access to others' users data. Access is granted to some data uploading tools but not all and some fields of updating forms are locked (see details of privileges in the document entitled S3_GOTIT_Help). Data export using the species search tools is possible either as .csv or .xls files.
- Users with the ROLE_PROJECT have CRUD rights on almost all data as well as access to most data uploading tools. Yet, some fields of updating forms remain locked.
- Users with the ROLE_ADMIN (administrator) have CRUD rights on all data, access to all uploading tools and all fields can be updated. Login details and privileges are granted to or revoked from the users by the administrator. The password encryption is done by the hash function sha512 (https://en.wikipedia.org/wiki/SHA-2).
- The access control of the login form is managed by Symfony (https://symfony.com/doc/3.4/security/form_login_setup.html)
- All forms implement the CSRF protection (https://symfony.com/doc/current/security/csrf.html)

I.6 – Bilingual interface

The web interface is produced in a bilingual version offering the possibility to shift between English and French at any time. Multilingualism is managed by the Symfony Translator service (https://symfony.com/doc/3.4/components/translation/usage.html) which uses dictionary files in yml format (https://symfony.com/doc/3.4/components/yaml/yaml_format.html). GOTIT has two dictionaries (messages and helps) that are located in the folder Resources/Translations of the core Bundle (E3sBundle). By default, the dictionaries are in two copies, one containing the French terms (fr) and the other the English terms (en). To set up the bilingual version so that it meets the user needs, the following parameters have to be modified:

- The _locale parameter of the routing.yml configuration file which specifies the two languages to be accepted in the url. Language codes are separated by a | without

spaces (for example, _locale: en|fr). A good practice is to use ISO 3166-1 alpha-2 for the choice of language codes.

- The local parameter (section parameters) of the config.yml configuration file which determines the default language of the login form and the interface (for example, locale: fr). It is recommended to select as default language the language used for terms stored in the vocabulary repository.
- The TWIG variable 'second_language' of the app / Resources / view / base.html.twig file sets the code of the second desired language for the interface, for example, *second_language = 'en'.*
- For a single-language version of the interface the parameters have to be adjusted as follows:
  (i) _locale: en
  (ii) local: en
  (iii) second_language = ''

While changing the languages used in the interface, you must rename the two dictionaries using the ISO codes of the desired languages. The French and English versions of the “helps” dictionary contain terms for the help elements of the interface (*helps.fr.yml and helps.en.yml*), whereas the “messages” dictionary contains terms for all other elements of the interface including buttons, menus and error messages. Changes to the content of these dictionaries should respect the following rules:

- The translations correspond to the texts on the right side of the two-point punctuation.
- Content on the left side of the two-point punctuation should not be changed.
- Files must be opened with a text editor that can read and write in UTF-8 format.
- Terms in the vocabulary repository table have to be translated into the second language in the message dictionary file. For the default language as set up in the configuration file, terms will be read directly from the database. Adding or modifying a term in the vocabulary repository table requires adding a translation of it and updating the dictionary files in the server.

I.7 – Software architecture & Third Party Developments

The GOTIT application is built according to the rationale of the Symfony framework. The file organization follows the MVC architecture with:

- The Information about the Model, that is to say the database structure, is in the E3sBundle/Entity folder.
- The user actions are stored in the E3sBundle/Controller folder. It contains all the controllers related to the different Use Cases of the application, that is to say the display of lists and CRUD actions (Create, Read, Update, Delete).

- HTML views are stored in the app/Resources/views folder. The names of the folders correspond to the names of the Routes defined in the controllers. Each individual folder includes TWIG templates that are all linked to the “base.html.twig” template (included at the root of the app/Resources/views folder).

The addition of third-party development should as far as possible promote the independence of the codes. To achieve this, the decision was made to implement third-party plugins as Symfony bundles (see https://symfony.com/doc/3.4/bundles.html for reference documentation). We use here the “Species search” bundle as a case example of an extension of the main application. This bundle implements specific interfaces to query species distributions and taxonomic concordance among different sets of species hypotheses.

This bundle was built as follows:

- Generate the bundle skeleton using Symfony command line tools:
php bin/console generate:bundle --
namespace=SomeNameSpace/ExampleNameBundle

Choose src/ as target directory and yml as configuration format.
This creates a directory src/SomeNameSpace/ExampleNameBundle with the described structure at:
https://symfony.com/doc/3.4/bundles.html#bundle-directory-structure
- Check that the files at AppKernel.php, routing.yml and config.yml in the app directory were updated. If not, update them with your bundle using the existing Species Search parameters in each file.
- Start creating pages as described in the Symfony documentation (https://symfony.com/doc/3.4/page_creation.html)
- Edit the template file at app/Resources/base.html.twig to add the navigation links to your new pages.
- Static assets, such as JS, CSS files or images, should be located at src/SomeNameSpace/ExampleNameBundle/Resources/public. After editing them, make them available in the /web/bundles/SomeNameSpaceExampleName/ directory using the following command :

```
php bin/console assets:install
```

- If you wish to add internationalization to your bundle, please refer to https://symfony.com/doc/3.4/translation.html

I.8 – Licence & legal notices

GOTIT application is released under the terms of the GNU GENERAL PUBLIC LICENSE, Version 3, dated at November 19, 2007

Legal notices are stored in the web/doc/ directory. You find the two files legal_notices_en.pdf and legal_notices_en.pdf relating to the bilingual interface en/fr. You must rename them and modify the contents according to the choice of languages.

To change the content of the login page with the institutions logos of your project, modify the login.html.twig template from App/Ressources/view/security/ directory. Copy all logos and images you need in the /web/e3s/images/ directory.

II – GOTIT installation tutorial

II.1 - Introduction

This tutorial explains how to install GOTIT project on your local system for managing and mapping data and metadata on species occurrence. To start you need to follow the following steps:

(i) Install and set up a web server Apache with PHP 7
(ii) Install the database management system PostgresSQL and set up the database
(iii) Install and set up the GOTIT web application
(iv) Start web server and connect to GOTIT

II.2 - Install and set up a web server Apache

The application is designed to work with an address of the type *http://localhost/gotit/*web . Copy the release of GOTIT in a folder entitled "gotit" under the www Web Apache directory. Then, install PHP in version 7.x (http://php.net/downloads.php) and the Apache web server. For installation, refer to the appropriate platform documentation (see https://httpd.apache.org/docs/2.4/en/platform/). Enable the Apache rewrite module (rewrite_module enabled), then edit the php.ini configuration file, and make changes as follows.

(i) Set the date.timezone (like UTC)
Set mbstring.func_overload to 0

(ii) Checking requirements for running Symfony:
test the configuration from the url like: *http://localhost/gotit/*web/config.php
see more details in https://symfony.com/doc/3.4/reference/requirements.html

(iii) Enable the upload of large .csv files (e.g. 5Mo max or about 25,000 lines max).
Enter the parameters:
file_uploads = On

upload_max_filesize = 5M
realpath_cache_size = 5M
max_execution_time = 300
max_input_time = 300
memory_limit = 512M
post_max_size = 8M

(iv) Enable PHP extensions:
php_pgsql , php_pdo_pgsql : for connecting to a postgreSQL database
php-xml : for managing DOM objects

II.3 - Install and set up the database management system PostgresSQL

a) Install the PostgreSQL database and the web client pgAdmin4

http://www.postgresqltutorial.com/install-postgresql/ (download postgreSQL v9.4)
https://www.pgadmin.org/download/

b) Create the tables in the db_gotit1 database

Use the script: dump_gotit1-1.sql
- Launch PgAdmin4 and create a connection to the server (localhost)
- Create a new database (db_gotit1) with UTF8 encoding
- Select the database and choose (right-click) « Query tool »
- Copy paste the dump_gotit1-1.sql script and click on the button Execute
- Note: this installation includes loading three postgreSQL extensions:
  (i) cube (https://www.postgresql.org/docs/9.4/cube.html)
  (ii) earthdistance (https://www.postgresql.org/docs/9.4/earthdistance.html)
  (iii) plpgsql (https://www.postgresql.org/docs/9.4/plpgsql.html)

c) Create the user management table

Use the script: db_user_gotit1-1.sql
- Start PgAdmin4 and select the database (db_gotit1)
- Select the database and choose (right-click) « Query tool »
- Copy paste the script db_user.sql, then click on the button Execute
- Note: the script loads a single administrator access account:
  login: admin
  password: adminGOTIT

II.4 - Install and set up the GOTIT web application

a) Download and install the last release of GOTIT

Go to the GOTIT GitHub page at https://github.com/GOTIT-DEV/GOTIT/tree/v1.1.1 and download the latest development sources from GitHub. Unzip the release somewhere in your web server's document root. You will see the following folders:

- app: all TWIG templates of views (/Ressources/views) and the configurations files (/config)
- bin: the symfony binary files
- src: include the two Bbees bundle : E3sBundle (the gotit application) and UserBundle (the user management application)
- tests: the repertory for Unit testing
- var: for symphony application temporary files (logs, cache …)
- vendor: all libraries and components of the framework (symfony, Doctrine …)
- web: include all other files: images, CSS, js, frontend controller, .htaccess, etc…
- See Appendix 1 for the list of all components used in GOTIT. The complete list of Symfony components are also available at https://symfony.com/components.

b) Connection to the database

Define access parameters to the postgreSQL database in the parameters.yml configuration file . First change name of file parameters.yml.dist in parameters.yml (see in ~/app/config directory) then enter the correct values for parameters like :

- database_host: 127.0.0.1
- database_port: 5432
- database_name: db_gotit1
- database_user: postgres
- database_password: *your_postgres_password*

c) Change footer and logos

You have to modify the content of footer in the “footer content” section in the TWIG templates (see in directory app/Resources/views) :

- secutity/login.html.twig
- base.html.twig

All logos and pictures should have to be stored in the /web/e3s/images web directory

II.5 - Start Web server and connect to GOTIT

To begin with GOTIT you have to:

- Start the Web server Apache and if necessary the postgresql server

- Connect to the login form with your browser: *http://localhost/gotit/*web/login

    login: admin

    password: adminGOTIT

- Change and create users profiles (“user” sub-menu), as you wish!
- Import the csv vocabulary file (vocabulary_gotit1-1.csv) to start with and add new terms to the parent list whenever necessary (“Repositories/Vocabulary” sub-menu + “import new set of Vocabularies” button).

Be careful, some content should not be changed!

(i) Titles of column (line 1)

(ii) The parent terminology (column C: “voc.parent”)

(iii) Labels of “datePrecision” and “leg” parents

(iv) Codes of “typeBoite” parent

APPENDIX 1 – List of GOTIT components

Component	Version	Action
composer/ca-bundle	1.1.0	Let's you find a path to the system CA bundle, and includes a fallback to the Mozilla CA bundle.
doctrine/annotations	1.2.7	Docblock Annotations Parser
doctrine/cache	1.6.2	Caching library offering an object-oriented API for many cache backends
doctrine/collections	1.3.0	Collections Abstraction library
doctrine/common	2.6.2	Common Library for Doctrine projects
doctrine/dbal	2.5.13	Database Abstraction Layer
doctrine/doctrine-bundle	1.8.1	Symfony DoctrineBundle
doctrine/doctrine-cache-bundle	1.3.2	Symfony Bundle for Doctrine Cache
doctrine/inflector	1.1.0	Common String Manipulations with regard to casing and singular/plural rules.
doctrine/instantiator	1.0.5	A small, lightweight utility to instantiate objects in PHP without invoking their constructors
doctrine/lexer	1.0.1	Base library for a lexer that can be used in Top-Down, Recursive Descent Parsers.
doctrine/orm	2.5.13	Object-Relational-Mapper for PHP
fig/link-util	1.0.0	Common utility implementations for HTTP links
friendsofsymfony/jsrouting-bundle	2.2.1	A pretty nice way to expose your Symfony2 routing to client applications.
incenteev/composer-parameter-handler	2.1.2	Composer script handling your ignored parameter file
jdorn/sql-formatter	1.2.17	a PHP SQL highlighting library
monolog/monolog	1.23.0	Sends your logs to files, sockets, inboxes, databases and various web services
paragonie/random_compat	2.0.11	PHP 5.x polyfill for random_bytes() and random_int() from PHP 7
psr/cache	1.0.1	Common interface for caching libraries
psr/container	1.0.0	Common Container Interface (PHP FIG PSR-11)
psr/link	1.0.0	Common interfaces for HTTP links
psr/log	1.0.2	Common interface for logging libraries
psr/simple-cache	1.0.0	Common interfaces for simple caching
sensio/distribution-bundle	5.0.21	Base bundle for Symfony Distributions
sensio/framework-extra-bundle	5.1.3	This bundle provides a way to configure your controllers with annotations
sensio/generator-bundle	3.1.6	This bundle generates code for you
sensiolabs/security-checker	4.1.6	A security checker for your composer.lock
swiftmailer/swiftmailer	5.4.8	Swiftmailer, free feature-rich PHP mailer
symfony/monolog-bundle	3.1.2	Symfony MonologBundle
symfony/phpunit-bridge	3.4.1	Symfony PHPUnit Bridge
symfony/polyfill-apcu	1.6.0	Symfony polyfill backporting apcu_* functions to lower PHP versions

symfony/polyfill-ctype	1.10.0	Symfony polyfill for ctype functions
symfony/polyfill-intl-icu	1.6.0	Symfony polyfill for intl's ICU-related data and classes
symfony/polyfill-mbstring	1.6.0	Symfony polyfill for the Mbstring extension
symfony/polyfill-php56	1.6.0	Symfony polyfill backporting some PHP 5.6+ features to lower PHP versions
symfony/polyfill-php70	1.6.0	Symfony polyfill backporting some PHP 7.0+ features to lower PHP versions
symfony/polyfill-util	1.6.0	Symfony utilities for portability of PHP codes
symfony/swiftmailer-bundle	2.6.7	Symfony SwiftmailerBundle
symfony/symfony	3.4.19	The Symfony PHP framework
twig/twig	1.35.0	Twig, the flexible, fast, and secure template language for PHP
willdurand/js-translation-bundle	2.6.6	A pretty nice way to expose your translation messages to your JavaScript.
willdurand/jsonp-callback-validator	1.1.0	JSONP callback validator.